②

Woods Hole
Oceanographic
Institution

Calculation of 3-Dimensional Synthetic Seismograms on the Connection Machine

by

J.M. Allen and D.R. Burns

October 1989

Technical Report

Funding was provided by the Office of Naval Research under Contract Numbers: N00014-87-K-0007 and N00014-89-J-1012.

2

WHOI-89-48

Calculation of 3-Dimensional Synthetic Seismograms on the Connection Machine

by

J.M. Allen and D.R. Burns

Woods Hole Oceanographic Institution
Woods Hole, Massachusetts 02543

October 1989

Technical Report

Funding provided by Office of Naval Research under
Contract Numbers: N00014-87-K-0007 and N00014-89-J-1012.

The Connection Machine operated by the Naval Research Laboratory Connection Machine Facility was used for part of this work. Another part of this work was conducted using the computational resources of the Northeast Parallel Architectures Center (NPAC) at Syracuse University, which is funded by and operates under contract to DARPA and the Air Force Systems Command, Rome Air Development Center (RADC), Griffiss Air Force Base, NY, under contract #F306002-88-C-0031.

Approved for Distribution:

[illegible]

Albert J. Williams 3rd, Chairman
Department of Applied Ocean Physics and Engineering

Contents

List of Figures

Abstract

A three dimensional, second order finite difference method was used to create synthetic seismograms for elastic wave propagation in heterogeneous media. These synthetic seismograms are used to model rough seafloor, the shallow crust, or complex structural and stratigraphic settings with strong lateral heterogeneities. The finite difference method is preferred because it allows models of any complexity to be generated and includes all multiple scattering, wave conversion and diffraction effects. The method uses a fully staggered grid as developed by Virieux (1986). Wavefront snapshots and time series output allow the scattering and focussing of different wave modes with direction to be visualized.

The extensive calculations required for realistic size models stretches the resources of serial computers like the VAX 8800. On the Connection Machine, a massively parallel computer, the finite difference grid can be directly mapped onto the virtual processors, reducing the nested time and space loops in the serial code to a single time loop. As a result, the computation time is reduced dramatically.

1 Introduction

Most computers used today are based on the von Neuman (serial) architecture. Serial computers have a single central processing unit (CPU) and memory; data from memory is passed to the CPU one data element at a time and instructions are executed in sequence. These computers are termed single-instruction, single-datastream (SISD) architecture since one instruction at a time is executed on a single data item. Although the efficiency of SISD computers may be enhanced by hardware techniques such as pipelining and vector processors, the time required to complete computationally intensive programs is often prohibitive.

Parallel systems can be grouped into two broad categories: single-instruction, multiple-datastream (SIMD) and multiple-instruction, multiple-datastream (MIMD). SIMD computers are characterized by a large number of simple processors, each with its own local memory connected by a network communications system. MIMD computers typically have a smaller number of conventional processors with shared memory which execute portions of a program concurrently.

In an application such as synthetic seismograms, which requires both large volumes of data and extensive calculations, the time required to cycle all the data through one CPU places severe limitations on the size of models which can be run on a serial computer. This report documents the procedure used to implement a software system to compute 3-dimensional synthetic seismograms by the finite differences method on the Connection Machine, a SIMD computer. The advantage of parallel processing on a SIMD computer is that each node of the 3D model can be assigned to a single processor. Calculations are then performed simultaneously on all grid points. An operation which would normally be performed within a repetitive loop is replaced by a single operation on many processors acting in parallel.

Two dimensional synthetic seismogram models were previously coded in FORTRAN 77 to run on VAX 11/780, VAX 8800, Cyber 205 and Cray XMP-12 computers (Hunt and Stephen, 1986). A FORTRAN 77 program for a three-dimensional solution to the elastic wave equation by the method of finite differences, which runs on the VAX 8800, was used as the basis for the parallel programs written in C*. The parallel program is called **wave_3d**.

1.1 The Connection Machine

The Connection Machine (CM) is a massively parallel computer developed by Thinking Machines, in Cambridge, MA. Massively parallel, or fine-grain computers have many simple processors, each with its own local memory. The Connection Machines used in this project are located at the Naval Research Laboratory (NRL) Connection Machine Facility and at the Northeast Parallel Architectures Center (NPAC) at Syracuse University. NRL has two CM-2 machines, one with 8k processors (Bambi) and one with 16k processors (Godzilla). NPAC also has two Connection Machines, a 32k CM-1 (CUBE) and a 32k CM-2 (SON-OF-CUBE).

The Connection Machine processors are divided into groups of 8k or 16k processors. A single Connection Machine can have one, two or four groups of processors. The 16k NRL CM-2 has two groups of 8k processors; the NPAC 32k CM-2 has 4 groups of 8k processors. Each group of processors is associated with a sequencer which interprets the instructions sent by a front-end computer. In order to use the Connection Machine system, the front-end must logically attach itself to one or more sequencers.

Each processor on the Connection Machine model CM-2 has 64K bits (2048 words) of memory; each CM-1 processor has 4k bits of memory. The Connection Machine processors are connected by a network communications system so that any processor can communicate with any other processor via a routing device wired in an n-cube pattern (Hillis,1987). The CM can be configured in software for virtual to physical processor ratios that are a power of 2, as long as there is enough memory per virtual processor to accomodate the required calculations and variables.

The memory requirements of the **wave_3d** program limits the maximum virtual processor ratio to 16:1. On the 16k NRL Connection Machine (Godzilla), the maximum number of virtual processors available for the 3D grid is 262,144 (64x64x64 or 256k); on the 32k NPAC CM-2 Connection Machine (SON-OF-CUBE), the maximum number of grid points is 524,288 (64x64x128 or 512k).

1.2 Programming the Connection Machine

The CM is connected by a high speed bus to a conventional serial computer which serves as the user-interface. Program development is accomplished on the front-end computer using the editors and compilers of the front-end processor. All of the Connection Machines used in this project support C*, C/Paris, *Lisp and CM Fortran.

The NRL Connection Machines Bambi (8k) and Godzilla (16k) are connected to four front-end computers:

1.	cmvax	(VAX 8800)
2.	cmsun	(SUN 4/280)
3.	THINK75	(Symbolics 3675)
4.	THINK40	(Symbolics 3640)

The NRL CMs are interfaced to the front-ends as follows:

Interface	cmvax	cmsun	THINK75	THINK40
0	Bambi	Bambi	Godzilla	Bambi
1	Godzilla	Godzilla		

The VAX front-end (cmvax) runs the ULTRIX operating system; the SUN front-end (cmsun) runs the UNIX operating system.

The two Connection Machines at NPAC (CUBE and SON-OF-CUBE) are connected to a VAX front-end (cmx) which runs ULTRIX 2.2, a 4.2BSD-based system.

The parallel program is downloaded onto the CM at runtime. The synthetic seismogram program, **wave_3d**, is written in C*, an extension of the C programming language, developed to allow parallel execution. The **wave_3d** program contains both serial and parallel code.

2 The Model

The finite difference method is used to create synthetic seismograms for models containing lateral heterogeneity; this method is preferred because it allows models of any complexity to be generated and includes all multiple scattering, wave conversion, and diffraction effects. Most analytical modelling techniques use assumptions, such as primary scattering (Born and Rytov approximations), which limit their applicability to weak heterogeneity situations. Modelling of a rough seafloor, the shallow crust, or complex structural and stratigraphic settings, however, must be able to handle strong lateral heterogeneities. In addition, the finite difference technique can be used for all ratios of scatterer size to wavelength.

2.1 Background

The finite difference method involves the spatial and temporal discretization of the wave equation on a regular grid which represents the model of interest. The method has been successfully applied to an extremely wide range of seismic wave propagation problems including earthquake seismology (Alterman and Karal, 1968; Frankel and Clayton, 1986; Toksoz et al., 1988), marine refraction (Stephen 1983; Dougherty and Stephen, 1988), reflection seismology (Kelly et al. 1976; Virieux, 1986; Fornberg, 1987), VSP (Stephen, 1984), and full waveform acoustic logging (Stephen et al., 1985). A number of formulations have been utilized for these applications, including second and fourth order formulations, and the pseudospectral method. Fornberg (1987) presents a comparison of these different methods. The higher order schemes are more accurate, require fewer grid points per wavelength, and are more computationally demanding. The lower order schemes are more efficient computationally, but require more grid points per wavelength, and therefore more computer memory. The second order scheme has been fully validated against analytic methods (Stephen, 1983, 1988) and can handle complicated interfaces and boundaries. In addition, the use of fully staggered grids for displacements and stresses (Virieux, 1986; Dougherty and Stephen, 1988) improves the accuracy and stability of the formulation at no additional cost in computation or memory. The second order scheme has been successfully applied to 2-D heterogeneous media by Dougherty and Stephen (1988), and 3-D media by Etgen and Yomagida (1988). Because the primary interest is in modelling rough interface effects, even the pseudospectral method will need fine grid spacing to accurately represent such interfaces. Therefore, the memory requirement differences between the lower order methods and the pseudospectral method are not as great as for other types of applications.

2.2 Approach

The system of equations we wish to solve is given by:

$$\rho\frac{\partial^2 u}{\partial t^2} = \frac{\partial \tau_{xx}}{\partial x} + \frac{\partial \tau_{xy}}{\partial y} + \frac{\partial \tau_{xz}}{\partial z}$$

$$\rho\frac{\partial^2 v}{\partial t^2} = \frac{\partial \tau_{xy}}{\partial x} + \frac{\partial \tau_{yy}}{\partial y} + \frac{\partial \tau_{yz}}{\partial z}$$

$$\rho\frac{\partial^2 W}{\partial t^2} = \frac{\partial \tau_{xz}}{\partial x} + \frac{\partial \tau_{yz}}{\partial y} + \frac{\partial \tau_{zz}}{\partial z}$$

$$\tau_{xx} = (\lambda + 2\mu)\frac{\partial u}{\partial x} + \lambda\frac{\partial v}{\partial y} + \lambda\frac{\partial w}{\partial z}$$

$$\tau_{yy} = \lambda\frac{\partial u}{\partial x} + (\lambda + 2\mu)\frac{\partial v}{\partial y} + \lambda\frac{\partial w}{\partial z}$$

$$\tau_{zz} = \lambda\frac{\partial u}{\partial x} + \lambda\frac{\partial v}{\partial y} + (\lambda + 2\mu)\frac{\partial w}{\partial z}$$

$$\tau_{xz} = \mu(\frac{\partial u}{\partial z} + \frac{\partial w}{\partial x})$$
$$\tau_{xy} = \mu(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x})$$
$$\tau_{yz} = \mu(\frac{\partial v}{\partial z} + \frac{\partial w}{\partial y})$$

Where u,v,w represent the x,y,z displacements, λ and μ are the Lame parameters, ρ is the density, τ_{ij} the stress terms, and t represents time. For heterogeneous media, the Lame parameters and density all vary spatially in three dimensions. A unique staggered spatial grid can be constructed to solve this system with centered differences. This grid is a direct extension of the two dimensional grid given in Virieux (1986).

In order to avoid grid dispersion in the model, the highest frequency component to be modelled must be sampled by 10 - 30 grid points per wavelength (Kelly et al., 1976; Stephen, 1983).

A dilatational or explosive point source is introduced into the grid using the method developed by Nicoletis (1981). This method computes a force distribution over a finite volume of the three dimensional grid to represent a dilatational source. The accuracy of this representation increases as the grid volume on which it is imposed is increased. The time dependency of the source takes the form of the derivative of the Gaussian distribution with any given center frequency (Kelly et al. 1976; Stephen et al., 1985).

Boundary conditions for this 3-dimensional staggered grid formulation are fairly straight forward. The source is introduced inside the model, and the six planes which define the model limits are treated as absorbing boundaries using the telegraph equation in a zone around each of the planes (Levander, 1985; Dougherty and Stephen, 1988). Because the finite difference model is formulated for heterogeneous media, interfaces within the model do not require any specific boundary conditions to be coded. Interfaces of any complexity are handled implicitly (with the caveat that steeply dipping interfaces must be adequately sampled by the spatial grid).

The memory and computational requirements for numerical modelling in three dimensions are great. For the second order three-dimensional code which we have developed, twelve (12) real variables must be stored for each grid point in the heterogeneous section of the model (three displacement values at three time steps, plus two elastic constants and density). In order to reduce the memory requirements on the VAX, the model assumes a heterogeneous zone sandwiched between two homogeneous zones. As an example of the memory and computational requirements on a serial machine, a model containing 80 x 80 x 50 grid points needs 12.5 Mbytes of central memory and the run time for 400 time steps was about 20 hours on a VAX 8800. These requirements can be reduced by reducing the heterogeneous zone to as small a volume as needed for the models of interest or by segmenting the model into several runs.

Fortunately, computer hardware advances in parallel processing can solve this problem. On a massively parallel processing machine the finite difference grid can be directly mapped onto the processors, reducing the nested time and space loops in the serial code to a single time loop in the parallel code. As a result, the computation time is reduced dramatically. Other parallel machines are designed with fewer more powerful processors. On these types of machines, each processor might run the finite difference code for a segment of the model, and communicate displacement values at the end of each time step. Any parallel computer can greatly reduce computation time and, therefore, make three dimensional numerical modelling feasible.

3 Running the synthetic seismogram program

This section describes the steps necessary to define the model, run the synthetic seismogram program and view the three-dimensional model created by the program **wave_3d**. Although some familiarity with the UNIX operating system is helpful, the following sections provide a complete description of the steps required.

To run program **wave_3d**:

1. Log on to the front-end via **telnet** (from the WHOI RED VAX)
2. Define the model parameters in the file **xmodel.dat**
3. Run the **wave_3d** program on the Connection Machine
4. View the results by either:
   (a) copying the ASCII output files to the VAX (via ftp) and using the SNAPSHOT program on the WHOI THISLE VAX Workstation, or
   (b) running **wave_3d** with the frame buffer option set and viewing the results on the CM Graphic Display System (GDS) high resolution color monitor (note that there should be someone at NRL or NPAC who can physically watch the frame buffer monitor!)

3.1 Logging on via telnet

Access to the Connection Machine front-end computers at NRL and NPAC is via the ARPA Internet link using **telnet** (figure 1).

1. log on to the WHOI Red VAX

   To access the Connection Machine at NRL:
   RED_$ telnet 134.207.7.12 (for the VAX interface - cmvax)
   RED_$ telnet 134.207.7.4 (for the SUN interface - cmsun)

   To access the Connection Machine at NPAC:
   RED_$ telnet cmx.npac.syr.edu (for the VAX interface - cmx)

   Note: when using **telnet**, you could get the following message:

   ```
   RED_$ telnet 134.207.7.12
   Trying...Attempt to connect to foreign host failed:  Network is unreachable
   RED_$ <<< at this point, just wait and try again later! >>>
   ```

2. log on to cmvax, cmsun or cmx; after logging on, one of the following prompts will appear:

```
cmvax%  (if logged on to the VAX front-end at NRL)
cmsun%  (if logged on to the SUN front-end at NRL)
cmx%    (if logged on to the VAX front-end at NPAC)
```

Figure 1a. Logging on to the NRL SUN front-end (cmsun)

```
RED_$ telnet 134.207.7.4

Trying... Open
Connected to 134.207.7.4.
Escape character is '^ ]'.

SunOS UNIX (cmsun)

login: allen
Password: <<<type password here >>>

Last login: Thu Aug 31 08:37:32 from RED.WHOI.EDU
SunOS Release 4.0 (CMSUN) #64: Tue Aug 22 12:45:06 EDT 1989

NRL Connection Machine Facility - Sun 4/280

<<< system messages here >>>
TERM = (vt100) <<< hit return here >>>
allen.cmsun% lo


Connection closed to 134.207.7.4

RED_$ <<< back to DCL prompt on RED VAX >>>
```

Figure 1b. Logging on to the NRL VAX front-end (cmvax).

```
RED_$ telnet 134.207.7.12

Trying... Open
Connected to 134.207.7.12.
Escape character is '^]'.

cmvax login: allen
Password: <<< type password here >>>

Last login: Tue Aug  8 11:31:48 from 128.128.16.79
Ultrix-32 V3.0 (Rev 64) System #7: Sat Jul  8 17:34:53 EDT 1989

NRL Connection Machine Facility - VAX 8800
    <<< system messages here >>>
TERM = (vt100) <<< hit return here >>>
allen.cmvax% lo


Connection closed to 134.207.7.12

RED_$ <<< back to DCL prompt on RED VAX >>>
```

Figure 1c. Logging on to the NPAC VAX front-end (cmx).

```
RED_$ telnet cmx.npac.syr.edu

Trying...
Open
Connected to cmx.npac.syr.edu.
Escape character is '^]'.


Ultrix-32 V3.0 (Rev 64) (cmx.npac.syr.edu)

login: jallen
Password: <<< type password here >>>

Last login: Tue Sep  5 09:17:39 from RED.WHOI.EDU
Ultrix-32 V3.0 (Rev 64) System #1: Mon Jul 17 11:24:59 EDT 1989
    <<< system messages here >>>
Tue Sep  5 09:17:40 EDT 1989

cmx% lo

Connection closed to cmx.npac.syr.edu
RED_$ <<< back to DCL prompt on RED VAX >>>
```

Figure 1: Example of logging on and off the NRL and NPAC Connection Machines

3.2 Defining the model input parameters

Input to program **wave_3d** is via the ASCII file **xmodel.dat**. The contents of **xmodel.dat** are as follows:

```
nt
dx,dy,dz,dt
snap_file,snap_fb
out_fb,out_xy,out_yz,out_xz
isx,isy,isz,fpeak
nzones
(for each zone:)
nx1,ny1,nz1,nxn,nyn,nzn
cclamb,ccmu,trho
```

where:

nt	= number of time steps
dx,dy,dz	= grid spacing in x,y,z directions, in meters
dt	= time step increment, in seconds
snap_file	= interval for output of ASCII files (number of time steps)
snap_fb	= interval for output to frame buffer (number of time steps)
out_fb	= 0 for no output to frame buffer
	= 1 to output xy plane to frame buffer, at snap_fb intervals
	= 2 to output yz plane to frame buffer, at snap_fb intervals
	= 3 to output xz plane to frame buffer, at snap_fb intervals
out_xy	= 1 to output xy divergence and curl z to ASCII files, at snap_file intervals
out_yz	= 1 to output yz divergence and curl x to ASCII files, at snap_file intervals
out_xz	= 1 to output xz divergence and curl y to ASCII files, at snap_file intervals
isx,isy,isz	= location of point source (grid points)
fpeak	= peak frequency of source (hz)
nzones	= number of homogeneous zones in model
nx1,ny1,nz1	= start point for zone in x,y,z directions (meters)
nxn,nyn,nzn	= end point for zone in x,y,z directions (meters)
cclamb	= Lame's parameter - lambda
ccmu	= Lame's parameter - mu (shear modulus)
trho	= density (kg/m)

Figure 2a shows a sample input data file for the model illustrated in figure 2b. Note that when defining the boundaries of the model areas, the indices (nx1,ny1,nz1,nxn,nyn,nzn) should range from 0 to 63, rather than from 1 to 64.

Use the **vi** editor on the front-end to edit this file or use the **edt** editor on the WHOI RED VAX and **ftp** the **xmodel.dat** file to the CM front-end (see section 3.4 for directions on transferring files via ftp).

Figure 2a. Sample file xmodel.dat for heterogeneous model.

This file describes the parameters for the model in figure 2b. The 3D grid is 64x64x64; the point source is located at (10,31,10); there will be no output to the frame buffer; ASCII files created will be: **div_xy.dat, curl_z.dat, div_xz.dat, curl_y.dat, div_yz.dat,** and **curl_x.dat.** The model will be calculated for 250 time steps and data will be output to the files at every 50 time steps.

```
250                      (number of time steps)
50 50 50 0.007           (dx, dy, dz, dt)
50 2                     (snap_files, snap_fb)
0 1 1 1                  (out_fb, out_xy, out_yz, out_xz)
10 31 10 3.              (isx, isy, isz, fpeak)
3                        (number of zones)
0 0 0 63 63 31           (nx1, ny1, nz1, nxn, nyn, nzn for zone 1)
2.25e09 0.0 1000.        (cclamb, ccmu, trho for zone 1)
0 0 32 31 63 63          (nx1, ny1, nz1, nxn, nyn, nzn for zone 2)
2.25e09 0.0 1000.        (cclamb, ccmu, trho for zone 2)
32 0 32 63 63 63         (nx1, ny1, nz1, nxn, nyn, nzn for zone 3)
16.0e09 8.0e09 2000.     (cclamb, ccmu, trho for zone 3)
```

Figure 2b. Three dimensional model calculated in the examples presented in this technical report.

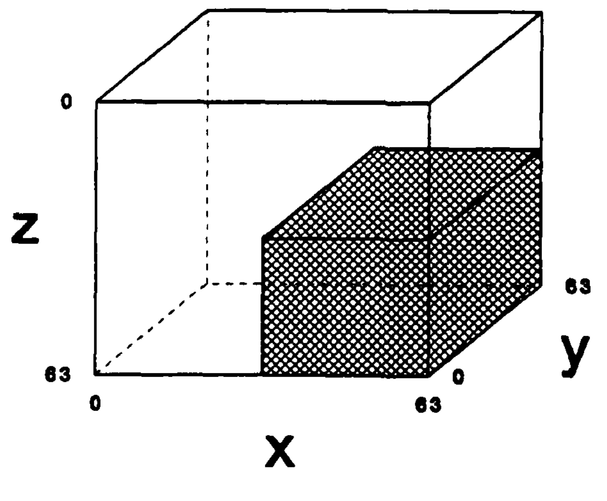


Figure 2: Sample input data file (xmodel.dat)

3.3 Running the program

Program **wave_3d** is configured for a model with dimensions 64x64x64; this model requires 262144 (256k) virtual processors. The program has been optimized to run at a maximum virtual processor ratio of 16:1. This means that a minimum of 16k processors is necessary to run this model. When running the program at NRL, all 16k processors of the Godzilla Connection Machine must be allocated. On the 32k NPAC CM, only two of the four sequencers on SON-OF-CUBE are needed to run the program. The program can also be configured to run a model with dimensions 64x128x64 on the NPAC 32k CM if all four sequencers on SON-OF-CUBE are used (see section 4.8).

To run the **wave_3d** program (figure 3):

1) Determine if the required sequencers on the CM are available using the cmfinger and cmuser commands (figure 4); you will not be allowed to attach to the CM if it is busy

2a) on the NRL CM, type the following command:

```
cmvax% cmattach -b 0.95 -p 16k -v 256k -i1 wave_3d > wave_3d.out &
```

2b) on the NPAC CM, use this command:

```
cmx% cmattach -b 0.95 -p 16k -v 256k -C S wave_3d > wave_3d.out &
```

The file **wave_3d.out** will contain the runtime diagnostic messages and the timing information after program completion. If the CM is busy and you wish to attach and 'wait for resources', add the -w flag to the command line:

```
cmvax% cmattach -b 0.95 -p 16k -v 256k -w -i1 wave_3d > wave_3d.out &
```

or

```
cmx% cmattach -b 0.95 -p 16k -v 256k -w -C S wave_3d > wave_3d.out &
```

This option will cause the cmattach program to wait (possibly forever) for the desired resources (namely both sequencers of Godzilla or two sequencers of SON-OF-CUBE) to be freed. Note that this is a fairly primitive implementation and simply causes the job to be resubmitted once every minute. If the sequencers are freed up during the minute that the job is waiting, they can be attached by anyone else. If you are in a hurry, it is safer, and usually faster, to use cmfinger to keep an eye on which sequencers are free and then resubmit the cmattach command. The c-shell has a history command that is useful for this:

cmx% !cma (the most recent command beginning with cma will be executed)

The output file should have a unique name on the NRL CM. If a file with that name exists already, the following message will appear:

```
cmsun% cmattach -b 0.95 -p 16k -v 256k -i1 wave_3d > wave_3d.out &
[1] 13999
wave_3d.out file exists
cmsun%
```

Either choose a different name for the output file on the command line, or remove the current file and re-submit the attach command:

```
cmsun% rm wave_3d.out
rm:  remove wave_3d.out? y
cmsun% !cma
```

On the NPAC CM, the file will be overwritten if it already exists.

To have diagnostic messages appear on the terminal screen rather than in a file, simply omit the redirection (> `wave_3d.out`):

```
cmsun% cmattach -b 0.95 -p 16k -v 256k -i1 wave_3d &
```

or

```
cmx% cmattach -b 0.95 -p 16k -v 256k -C S wave_3d &
```

3.4 Viewing the results

In order to view the results of the 3-dimensional synthetic seismogram program, 2-dimensional 'slices' through the center of the model may be:

1. output as ASCII files and transferred to a VAXstation for plotting, or
2. output to the Connection Machine frame buffer during runtime.

3.4.1 *Plotting ASCII files on the VAXstation*

Program **wave_3d** writes two ASCII files (one containing divergence and the other containing curl) for each 2-dimensional plane selected, every time step interval. The filenames are determined by the 2-dimensional plane and the time step:

output option	files	description
out_xy	divxy_N.dat	xy plane at z = z-dimension/2-1
	curlz_N.dat	
out_yz	divyz_N.dat	yz plane at x = x-dimension/2-1
	curlx_N.dat	
out_xz	divxz_N.dat	xz plane at y = y-dimension/2-1
	curly_N.dat	

where N is the time step.

The procedure for plotting ASCII files of divergence and curl on the VAXstation is as follows:

1. Transfer the ASCII files from the CM front-end to the RED VAX using ftp:
   ```
   RED_$ ftp 134.207.7.12 (refer to section 4.4)
   username: allen
   password: <<type password here>>
   * get divxz_200.dat (to transfer divxz_200.dat from cmvax to RED)
   * get curly_200.dat (to transfer curly_200.dat from cmvax to RED)
   * ex (to exit ftp)
   ```

Figure 3a. Running wave_3d on the NRL cmsun front-end

```
cmsun% cmfinger<<< make sure that both sequencers on GODZILLA are free >>>

Connection Machine: BAMBI

        nobody          CMSUN:0                 Not attached.
        LISPM           THINK75                 Not attached.
        nobody          CMVAX:0                 Not attached.

Connection Machine: GODZILLA

        nobody          CMSUN:1                 Not attached.
        webb            THINK40                 Not attached.
        nobody          CMVAX:1                 Not attached.

cmsun% cmattach -b 0.95 -p 16k -v 256k -i1 wave_3d > wave_3d.out &
[1] 990
cmsun% Attaching the Connection Machine system GODZILLA... cold booting...done.
Attached to 16384 processors, on sequencers 0 and 1, microcode version 5110
95.00% of memory is reserved for back compatibility mode.
262144 virtual processors were configured (2048 x 128).
There are 3796 bits of user memory per virtual processor.
Paris safety is off.
cmsun% cmfinger
Connection Machine: BAMBI

        nobody          CMSUN:0                 Not attached.
        LISPM           THINK75                 Not attached.
        nobody          CMVAX:0                 Not attached.

Connection Machine: GODZILLA

        allen           CMSUN:1                 Sequencer ports (0 1)
        webb            THINK40                 Not attached.
        nobody          CMVAX:1                 Not attached.
cmsun% cmusers
Interface  User      CM-Idle  Status
    1      allen              ATTACHED running "wave_3d"

cmsun% Detaching... done.
[1]    Done         cmattach -b 0.95 -p 16k -v 256k -i1 wave_3d > wave_3d.out
cmsun%
```

Figure 3b. Running wave_3d on the NPAC cmx front-end

```
cmx% cmfinger <<< make sure that two adjacent sequencers are free >>>
Connection Machine: THE-CUBE

        nobody          CMX.NPAC.SYR.EDU:0   Not attached.

Connection Machine: SON-OF-CUBE

        nobody          CMX.NPAC.SYR.EDU:1   Not attached.
        nobody          CMX.NPAC.SYR.EDU:2   Not attached.
        nobody          CMX.NPAC.SYR.EDU:3   Not attached.
        CM1S                                 Not attached.
cmx% cmattach -b 0.95 -p 16k -v 256k -C S wave_3d ¿ wave_3d.out &
[1] 6819
cmx% Attaching the Connection Machine system SON-OF-CUBE... cold booting...done.
Attached to 16384 processors on sequencers 0 and 1, microcode version 5110
95.00% of memory is reserved for back compatibility mode.
262144 virtual processors were configured (2048 x 128).
There are 3796 bits of user memory per virtual processor.
Paris safety is off.
cmx% cmfinger
Connection Machine: THE-CUBE

        nobody          CMX.NPAC.SYR.EDU:0   Not attached.

Connection Machine: SON-OF-CUBE

        jallen          CMX.NPAC.SYR.EDU:1   Sequencer ports (0 1)
        nobody          CMX.NPAC.SYR.EDU:2   Not attached.
        nobody          CMX.NPAC.SYR.EDU:3   Not attached.
        dgr             CM1S                 Not attached.
cmx% cmusers
Interface  User       CM-Idle  Status
    1      jallen              ATTACHED running "wave_3d"

cmx% Detaching... done.

[1]  Exit -46 cmattach -b 0.95 -p 16k -v 256k -C S wave_3d > wave_3d.out

cmx%
```

Figure 3: Example of cmattach command

Figure 4a - Example of cmfinger and cmusers commands on the NRL cmvax front-end

```
cmvax% cmfinger

Connection Machine: BAMBI

        nobody        CMSUN:0            Not attached.
                      THINK75            Not attached.
        whitcomb      CMVAX:0            Sequencer ports (0)

Connection Machine: GODZILLA

        ancona        CMSUN:1            Sequencer ports (1)
        webb          THINK40            Not attached.
        jennings      CMVAX:1            Not attached.

cmvax% cmusers

Interface  User      CM-Idle  Status
   0       whitcomb           ATTACHED running "starlisp"
   1       jennings    :03    ATTACHED running "cmattach"
cmvax%
```

Figure 4b - Example of cmfinger and cmusers commands on the NRL cmvax front-end

```
cmsun% cmfinger

Connection Machine: BAMBI

        tedwards      CMSUN:0            Sequencer ports (1)
                      THINK75            Not attached.
        mandelbe      CMVAX:0            Sequencer ports (0)

Connection Machine: GODZILLA

        allen         CMSUN:1            Sequencer ports (0 1)
        webb          THINK40            Not attached.
        nobody        CMVAX:1            Not attached.

cmsun% cmusers

Interface  User      CM-Idle  Status
   0       tedwards           ATTACHED running "cmattach"
   1       allen              ATTACHED running "wave_3d"
   -       root        :47    DETACHED
cmsun%
```

Figure 4c - Example of cmfinger and cmusers commands on the NPAC cmx front-end

```
cmx% cmfinger

Connection Machine: THE-CUBE

        nobody        CMX.NPAC.SYR.EDU:0 Not attached.

Connection Machine: SON-OF-CUBE

        nobody        CMX.NPAC.SYR.EDU:1 Not attached.
        root          CMX.NPAC.SYR.EDU:2 Sequencer ports (3)
        nobody        CMX.NPAC.SYR.EDU:3 Not attached.
                      CM1S               Not attached.

cmx% cmusers

Interface  User      CM-Idle  Status
   2       root       1:52    ATTACHED running "1"

cmx%
```

Figure 4: Example of cmfinger and cmusers commands

```
RED_$
```

2. On the RED VAX, convert the ASCII output files to binary format using the convert program:

```
RED_$ @convert
Enter CM (input) filename: divxz_200.dat
Enter binary output filename: cmxz0200.div
Enter scale factor (1 for no scaling): 1
FORTRAN STOP
RED_$ @convert
Enter CM (input) filename: curly_200.dat
Enter binary output filename: cmxz0200.crl
Enter scale factor (1 for no scaling): 1
FORTRAN STOP
RED_$
```

3. Log onto the VAX Workstation THISLE (THISLE is on the VAX cluster so the default data directory RNR2:[FIND.JMA1] can be accessed from either RED or THISLE).

4. On THISLE, run **snap_fix** to enlarge the images, if desired (this example will enlarge a 64x64 image to 256x256):

```
THISLE_$ r snap_fix
Enter input filename (fin): cmxz0200.div
Enter output filename (fout): cbxz0200.div
Enter x,z dimensions (max 128): 64,64
Enter xinx,zinc: 4,4
FORTRAN STOP
THISLE_$
```

5. On THISLE, execute the command SNAP to send the image to the workstation screen and/or create an image file. The example in figure 5 creates an image file named **cbxz0200.img**.

6. To get a hardcopy of the image file(s) created by SNAP, log back onto the RED VAX and run the command file `img`. Output is to the Imagen laser printer (figure 6):

```
RED_$ @img cbxz0200.img
```

3.4.2 Using the CM frame buffer

The CM graphics display system (GDS) consists of a frame buffer and a high-resolution color monitor (19 inch, 1280 x 1024 pixels), which combine to display images computed on the CM-2. The GDS can be programmed to display any 2-dimensional grid of virtual processors.

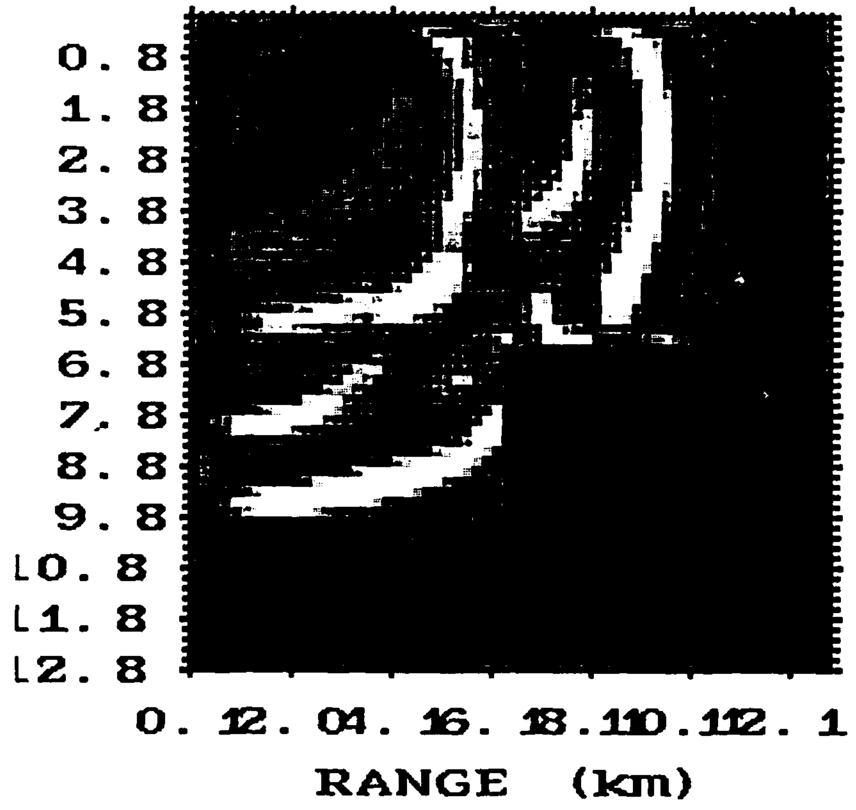
If the variable *out_fb* is set to 1, 2 or 3 in the file **xmodel.dat**, images will appear on the default frame buffer at NRL or NPAC in 'real-time' (this is obviously only useful if there is someone at NRL or NPAC who is watching the monitor!).

At NRL, the four frame buffer locations are as follows:

<<< note that caps are required for portions of this program >>>

```
THISLE_$ SNAP
ENTER FILEID
CBXZ
ISTAT=           0
ERROR   29  OPENING FILE CBXZ.PAR
ISTAT=          29
ENTER MM,NN,ND,DELZ
64,64,0,50
ENTER TIMESTEP
200
ENTER TOP FILE EXTENT
DIV
ENTER BOTTOM FILE EXTENT ("N" FOR NO)
CRL
FILE OPENED:  CBXZ0200.DIV
FILE OPENED:  CBXZ0200.CRL
ENTER REDUCE FACTOR
1
         1
ENTER RANGE MIN,MAX (POINTS)
1,64
         1          64
ENTER DEPTH MIN,MAX (POINTS)
1,64
         1          64
ENTER CLIPPING MIN,MAX VALUES
-0.01,0.01
CLIPPING VALUES=  -9.9999998E-03  9.9999998E-03
ENTER ANNOTATION FOR TITLE
<<<  hit return here  >>>
OUPUT TO WORKSTATION MONITOR? (Y/N)
Y  <<<  use the workstation mouse to manipulate the menus  >>>
NEW COLOR MAP ? (Y/N)
N
RESCALE ? (Y/N)
N
OUTPUT IMAGE ? (Y/N)
Y
FORTRAN STOP
THISLE_$
```

Figure 5: Example of running SNAP on THISLE



Section along the x-z plane at time step 200 of 3-dimensional synthetic seismogram calculated with program wave_3d. The model calculated is shown in figure 2. Image displayed on microVAX workstation using program SNAPSHOT.

Figure 6: Hardcopy image on the laser printer

CM	interface	sequencer	FB location
Bambi	0	0	User room
Bambi	0	1	Anderson/Whaley office
Godzilla	1	0	Room with coffee maker/fridge
Godzilla	1	1	Shirron's office

At NPAC, the frame buffers are attached to sequencers 0 and 2 of SON-OF-CUBE.

4 Guide to programming on the Connection Machine

The first step in writing a program to run on the Connection Machine is to learn to think in parallel terms. "Learning to write programs for parallel machines requires thinking in ways that are quite different from those demanded by sequential computers." (Hillis, 1987)

4.1 Special cases and boundary conditions

The large number of processors in the Connection Machine are controlled by the front-end computer, which sends a single instruction to all the processors simultaneously. For every instruction, each processor must either execute or not execute. Special cases and boundary conditions are one weakness of parallel processing. Algorithms which contain special cases should be redesigned, if possible.

For example, consider the calculation of boundary conditions. On a serial machine, the interior of the model would be calculated with one equation and the edges of the model with another. On the Connection Machine, however, it is most efficient for every processor to perform the same equation simultaneously. One solution is to add a damping term to the wave equation to eliminate unwanted reflections from the edges of the model. Then the same equation can be used for every processor with only the damping parameter changing with position on the grid (Charrette, 1987).

4.2 Timing considerations

Another power of programming on a massively parallel computer such as the Connection Machine is that increasing the size of the model (within the hardware limits of the computer) will not change the time required to perform the parallel calculations, although serial operations such as initialization will take longer. For example, **wave_3d**, the 3D synthetic seismogram program, has three main sections:

1. code to read model parameters and perform parallel initialization
2. code to perform parallel calculations within a serial time loop
3. serial code for output of results

It is difficult to make an accurate comparison of runtimes on the CM and runtimes on the VAX 8800. The timing facility on the CM gives real time and CM time (time spent in parallel mode) while the VAX timing facility gives real time and CPU time. Comparison of real times is not entirely valid because of multi-tasking on the computers. In this report, we are comparing real time on the Connection Machine with CPU time on the VAX; the performance of the Connection Machine is actually under-estimated.

A 3D model with dimensions 64x128x64 and 200 time steps took 0.5 seconds (real time) for initialization and 1253.6 seconds (real time) total on the 32k Connection Machine at NPAC.

A 3D model with dimensions 80x80x50 and 200 time steps took 400 minutes (CPU time) on the VAX 8800. The following table summarizes the timing calculations:

Computer	Number of grid points	seconds / time step
CM-2 (32k)	524288 (64x128x64)	6.3 (real time)
VAX 8800	320000 (80x80x50)	120 (CPU time)

The Connection Machine is approximately twenty times faster than the VAX for a model with 60% more points. Although models computed on the Connection Machine are constrained by the number of virtual processors available, it may be possible to compute larger models using the DataVault, a high speed mass storage device attached to the Connection Machine (see section 6.0).

4.3 Converting serial FORTRAN to parallel C*

This section outlines some special features of C* and some guidelines to assist a programmer in developing and modifying C* code for the Connection Machine.

The original 3-dimensional synthetic seismogram program was written in FORTRAN and developed to run on the VAX, a serial computer. The conversion of the serial FORTRAN code to parallel C* code was accomplished in four basic steps:

1. convert from FORTRAN to standard C
2. determine portions of the code which could be run in parallel
3. convert from standard C to parallel C*
4. optimize the C* code

4.3.1 Convert from FORTRAN to standard C

One of the most common coding errors in converting from FORTRAN to C is in indexing. FORTRAN indexes from 1 to N while C indexes from 0 to N-1. For example, a source located at the center of a 64x64x64 model would have indices (32,32,32) in FORTRAN while the source located at the center of the same model would have indices (31,31,31) in C.

4.3.2 Determine portions of the code which could be run in parallel

The C* programming language was designed specifically for the Connection Machine. C* is an extension of the C programming language that incorporates some of the object-oriented features of C++ to enable parallel processing.

The C* language encourages the programmer to think in terms of groups of related quantities, similar to the *struct* concept in standard C. In C*, the keyword *domain* extends the *struct* concept by allowing functions, as well as data objects, to be associated members of a group. *Domains* are based on the *class* concept from C++. Once a *domain* is defined and member space is allocated, each instance of the *domain* can be thought of as having its own virtual processor. A program can contain both serial and parallel code; code is parallel only within a member function of a *domain* or within a selection statement which has activated a *domain*. Otherwise, the parallel code appears just like serial code.

Parallel variables are the default when declared within a member function and within a selection statement and can be explicitly declared with the *poly* keyword within serial code. Serial variables require the keyword *mono* if declared within a parallel context.

Many of the computations in the original synthetic seismogram program involved large, 3-dimensional FORTRAN *do loops* where the same computation was performed for each node of the 3-dimensional model. These computations are relatively easy to think of in a parallel sense. Instead of processing each node of the model in sequence, all nodes are processed in parallel by placing the same code on each of the parallel processors. The first step is to allocate a virtual processor for each grid point using the nd_grid library routine *make_grid* (see Appendix A). A *domain* with 23 common variables and 20 parallel functions was defined for the 3D model (Figure 7). Once the parallel *domain* is defined, operations which require a 3-dimensional *do loop* in FORTRAN are reduced to a single operation in C* (Figure 8).

4.3.3 Convert from standard C to parallel C*

The following C* header files are available for parallel C* programs:

<stdio.hs>	standard i/o library header file
<math.hs>	math routines header file
<nd_grid.hs>	header for n-dimensional NEWS routines
<fb.hs>	header for frame buffer routines
<cm/cmtimer.hs>	CM timer functions
<cm/cmfs.hs>	CM file structure (DataVault) functions
<cm/cm_file.hs>	DataVault functions

4.3.4 Optimize the C* code

Each processor on the CM-2 has 64k bits of memory and this was found to be the major constraint on the maximum size model that could be run. Techniques used to decrease the amount of memory required by the program and maximize model size include:

1. re-compute values rather than store them in memory
2. break up large functions into smaller functions in order to reduce the amount of memory required

There is a 'bug' in the C* compiler which causes any floating point constant which is not implicitly appended with an *f* to be double precision. This causes any calculations using that constant to be

```
/* common definitions for 3D finite difference modeling */
#define         XDIM            64
#define         YDIM            64
#define         ZDIM            64
#define         TOTAL           XDIM*YDIM*ZDIM

/* define the domain for the parallel variables - 27 variables = 108
bytes */
domain model {
        float u0,u1,u2;
        float v0,v1,v2;
        float w0,w1,w2;
        float clamb;
        float cmu;
        float clp2mu;
        float rho;
        float txxdx,txydy,txzdz;
        float txydx,tyydy,tyzdz;
        float txzdx,tyzdy,tzzdz;
        float alpha;
        float div,curlx,curly,curlz;
        float alpha_fct(int indx);
        void boundary(void);
        void calc_adisp(float *adisp);
        void calc_bdisp(float *bdisp);
        void calc_cdisp(float *cdisp);
        void calc_displ(void);
        void calc_txxdx(void);
        void calc_txydy(void);
        void calc_txzdz(void);
        void calc_txydx(void);
        void calc_tyydy(void);
        void calc_tyzdz(void);
        void calc_txzdx(void);
        void calc_tyzdy(void);
        void calc_tzzdz(void);
        void divcurl(void);
        void initialize(void);
        void output_fb(void);
        void stress(void);
        void update(void);
} point[TOTAL];
```

Figure 7: Declaration of domain for 3D synthetic seismograms

```
c===================================================================
c       FORTRAN subroutine update - update arrays after completion
c               of a time step
c===================================================================
        subroutine update
c
        include 'dspall.blk'
        common/values/ ltmout,ixout,iyout,izout,nx,ny,nz,nt,nt1
c
        do 300 iz = 1, nz
            do 200 iy = 1, ny
                do 100 ix = 1, nx
                    uold(ix,iy,iz) = u(ix,iy,iz)
                    vold(ix,iy,iz) = v(ix,iy,iz)
                    wold(ix,iy,iz) = w(ix,iy,iz)
c
                    u(ix,iy,iz) = unew(ix,iy,iz)
                    v(ix,iy,iz) = vnew(ix,iy,iz)
                    w(ix,iy,iz) = wnew(ix,iy,iz)
100             continue
200         continue
300     continue
        return
        end
c===================================================================

/*================================================================*/
/*                                                                */
/*      C* function update - update arrays after completion of    */
/*            a time step                                         */
/*                                                                */
/*================================================================*/
#include "wave_3d.hs"
void model::update(void)         /* function type is domain model */
{
    uold = u;
    vold = v;
    wold = w;

    u = unew;
    v = vnew;
    w = wnew;
}
/*================================================================*/
```

Figure 8: Example of serial FORTRAN vs parallel C*

done in double precision, thus wasting valuable memory resources. To avoid this problem, append an *f* to all floating point constants in the C* program, for example:

`#define PI 3.14159f` (instead of `#define PI 3.14159`)

```
alpha = sin(((ix-(nx-NTEL))/(NTEL/2.0f))*PI/2.0f)*(0.1f);
```

To detect lines containing values which would cause double precision computations, use the -w flag with the compile command:

```
cmx% cs -O -w -o progname progname.cs -lnrlcstar &
```

There is a timing facility in C* for accumulating and reporting elapsed real time and CM time. To use the timing facility, include the header file:

```
#include <cm/cmtimer.hs>
```

The timer functions used in the current program are:

CM_start_timer(1)	begin accumulating timing information
CM_stop_timer(1)	stop accumulating and print timing information
CM_reset_timer()	reset the clock

These functions were used to time the initialization stage of the program and the total run-time. The timer functions can be implemented as C preprocessor macros as follows:

```
#define TIMER_ON CM_start_timer(1)

#define TIMER_RESET CM_reset_timer()

#define TIMER_OFF CM_stop_timer(1)
```

4.4 File transfer via ftp

The internet connection to NRL is very slow. For this reason, most of the program coding and editing was done on the Red VAX at WHOI. Programs were then transferred to the CM front-end over the network using file transfer protocol (FTP), illustrated in Figure 9.

1. ftp to connect to the CM front-end
   ```
   RED_$ ftp 134.207.7.12 <<< to connect to the NRL VAX >>>
   RED_$ ftp 134.207.7.4 <<< to connect to the NRL SUN >>>
   RED_$ ftp cmx.npac.syr.edu <<< to connect to the NPAC VAX >>>
   cmx% ftp red.whoi.edu <<< to connect to the WHOI RED VAX >>>
   ```

2. cd to the desired directory if files are in a directory other than HOME

3. use **get** to copy files FROM the CM front-end, use **put** to copy files TO the CM front-end

4. **ex** to exit ftp from the WHOI VAX, (**bye** to exit from the UNIX front-ends)

4.5 Debugging C* Code

Debugging the parallel portions of a C* program can be difficult. It is generally best to start with a simple model and print out intermediate values for selected processors. If a printf statement is used within a parallel context, it must be immediately followed with:

```
fflush(stdout);
```

This will ensure that all buffered data is sent to the output stream before parallel commands are executed.

Section 5.3.3 of the Connection Machine System C* User's Guide explains how to run C* programs with the *dbx* debugger.

4.6 Compile and link CM programs

Source code to be run on the CM should have the file extent *.cs* and header files should have the extent *.hs*. Figure 10 illustrates compilation of **wave_3d** on the NRL and NPAC front-end computers.

To compile and link a C* program to run on the NRL Connection Machine (the NRL C* library is installed as nrlcstar):

```
cmvax% cs -O -w -o progname progname.cs -lnrlcstar &
```

where:

-O	sets the flag for the optimizer
-w	causes double precision calculations to be flagged
-o progname	causes the executable file to be named progname
instead of a.out	
progname.cs	is the C* source code file
-lnrlcstar	links the program with the NRL C* library
-lcmfb	links the program with the frame buffer library
&	the UNIX command for running in the background

To compile and link a C* program to run on the NPAC Connection Machine (the NRL C* library routines are local):

```
cmx% cs -O -w -o progname progname.cs nd_grid.o fb.o &
```

4.7 Running a C* program on the Connection Machine

To run a program on the NRL Connection Machine:

<u>Figure 9a. Sending a file to NRL cmsun using file transfer protocol (ftp)</u>

```
RED_$ ftp 134.207.7.4

red Wollongong FTP User Process (Version 5.0) Connection Opened
 Using 8-bit bytes.
220 cmsun FTP server (SunOS 4.0) ready.
Name (134.207.7.4:ipc_jma1): allen

331 Password required for allen.
Password: <<< enter password >>>

230 User allen logged in.
* put wave_3d.cs

200 PORT command successful.
150 ASCII data connection for wave_3d.cs (128.128.16.2,1569).
226 Transfer complete.
47628 bytes transfered in 32.26 seconds (1.44 Kbytes/second)
* put wave_3d.hs

200 PORT command successful.
150 ASCII data connection for wave_3d.hs (128.128.16.2,1570).
226 Transfer complete.
2847 bytes transfered in 0.01 seconds (278.03 Kbytes/second)
* ex

221 Goodbye.
RED_$
```

<u>Figure 9b. Getting a file from NPAC cmx using file transfer protocol (ftp)</u>

```
RED_$ ftp cmx.npac.syr.edu

red Wollongong FTP User Process (Version 5.0)
 Connection Opened
 Using 8-bit bytes.
220 cmx.npac.syr.edu FTP server (Ultrix Version 4.10 Mon Nov 7 15:52:11 EST 1988
) ready.
Name (cmx.npac.syr.edu:ipc_jma1): jallen

331 Password required for jallen.
Password: <<< enter password >>>

230 User jallen logged in.
* cd findif
250 CWD command successful
* get wave_3d.out

200 PORT command successful.
150 Opening data connection for wave_3d.out (128.128.16.2,1263) (6747 bytes).
226 Transfer complete.
6618 bytes transferred in 1.24 seconds (5.21 Kbytes/second)
* ex

221 Goodbye.
RED_$
```

Figure 9: Example of using file transfer protocol (ftp)

Figure 10a - Example of compiling wave_3d on the NRL cmsun front-end

```
cmsun% cs -O -w -o wave_3d wave_3d.cs -lnrlcstar &

[1] 18467
wave_3d.cs: C*-compiling => wave_3d..c
C* Compiler Version 5.0.21
"wave_3d.cs", line 98: warning: Underflow from floating-point literal 0.0.
"wave_3d.cs", line 98: warning: Underflow from floating-point literal 0.0.
"wave_3d.cs", line 100: warning: Underflow from floating-point literal 0.0.
"wave_3d.cs", line 977: warning: Underflow from floating-point literal 0.0.
wave_3d.cs: C-compiling wave_3d..c
Linking
[1]    Done                 cs -O -w -o wave_3d wave_3d.cs -lnrlcstar
cmsun%
```

Figure 10b - Example of compiling wave_3d on the NPAC cmx front-end

```
cmx% cs -O -w -o wave_3d wave_3d.cs nd_grid.o fb.o &

[1] 5996
cmx% wave_3d.cs: C*-compiling => wave_3d..c
C* Compiler Version 5.0
wave_3d.cs: C-compiling wave_3d..c
Linking

[1]    Done            cs -O -w -o wave_3d wave_3d.cs nd_grid.o fb.o
cmx%
```

Figure 10: Compilation of wave_3d

```
cmsun% cmattach -b 0.95 -p nprocs -v nvprocs -i interface progname > output_file &
```

To run a program on the NPAC Connection Machine:

```
cmx% cmattach -b 0.95 -v nvprocs -p nprocs -C CM_name progname > output_file &
```

where:

-b 0.95	causes 95% of the memory per virtual processor (vp) to be available (default is 0.75)
-p nprocs	attach nprocs CM processors; legal values for nprocs: 4k or 8k for BAMBI, (default is 4k) 8k or 16k for GODZILLA (default 8k) 8k, 16k or 32k for SON-OF-CUBE (default 8k)
-v nvprocs	configure the CM to have nvprocs virtual processors (must be at least the number of physical processors allocated)
-i interface	attaches the CM via interface 0 (BAMBI) or interface 1 (GODZILLA) on NRL CM
-C CM_name	attaches to the CM named 'CM_name' (use -C S for SON-OF-CUBE at NPAC)
progname	your executable C* program
output_file	file containing log of batch session
&	causes program to run in the background

The Connection Machine System C* User's Guide (section 4.3) includes instructions for executing C* programs interactively.

4.8 Re-configuring wave_3d

The **wave_3d** program is currently configured for models with dimensions 64x64x64, which is the largest model that will run on a 16k Connection Machine. The 32k Connection Machine at NPAC can accomodate a model with dimensions 64x128x64. To change the model dimensions in the **wave_3d** program:

1. edit the header file **wave_3d.hs** and modify any or all of the following lines (note that model dimensions MUST be a power of 2!):

```
#define      XDIM      64
#define      YDIM      64
#define      ZDIM      64
```

2. re-compile the **wave_3d** program (refer to section 4.6)

```
cmx% cs -O -w -o wave_3d wave_3d.cs nd_grid.o fb.o &
```

3. edit **xmodel.dat** to incorporate the new model parameters

4. attach and run the program, making the appropriate changes to the cmattach parameters (refer to section 4.7)

   cmx% cmattach -b 0.95 -p 32k -v 512k -C S wave_3d > wave_3d.out &

5 Frame Buffer

The Connection Machine graphic display system consists of a framebuffer and a high resolution color monitor. The framebuffer contains a display memory capable of storing an image 2048 by 1024 pixels with three eight-bit color buffers and a four-bit overlay buffer. The color monitor has a resolution of 1280 by 1024 pixels. Since the display memory can store more pixels than the monitor can display, pan and zoom operations are supported.

The CM graphic display system allows each virtual processor to be mapped to a pixel on the color monitor. As values from the virtual processors are written to the display memory, the image on the monitor changes. For the 3D synthetic seismograms, a 2-dimensional plane is selected (for example, the xz plane) and 'slices' through the model are displayed at selected time steps in 'real-time'.

The framebuffer is normally accessed via C/PARIS (Connection Machine PARallel Instruction Set) calls. The **wave_3d** program uses the NRL C* library fb which is a collection of functions used to display pixels on the framebuffer (see Appendix A). To use these functions in a C* program, include the header file:

```
#include "fb.cs"
```

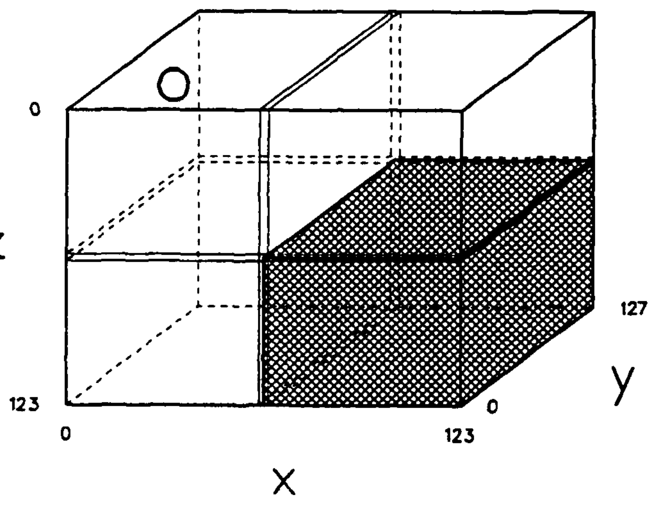
and link with the library: `-lnrlcstar` (if using NRL Connection Machine) or `fb.o` (if using NPAC Connection Machine).

6 DataVault

The DataVault is a 5-Gigabyte mass storage system which provides a data parallel file system for the Connection Machine. Data can be transferred to and from the CM-2 in parallel at high speed (40 megabytes per second) or it can be read and written directly from the front-end without using the CM-2.

The amount of memory per processor and the number of physical processors on the Connection Machine constrains the maximum size of the 3D model that can be computed.

In order to calculate 3D synthetic seismograms for models with dimensions of 128x128x128 (actual dimensions 124x128x124 with overlap), the model can be divided into four segments with an overlap of 4 grid points (figure 11). Each segment can be copied to a temporary file on the DataVault at each time step. Preliminary calculations indicate that approximately 2 seconds per time step are required to write the 23 parallel *domain* variables to four temporary files on the DataVault.



Three-dimensional model divided into four segments; each segment has dimensions 64x128x64. The segments overlap four grid points in the x- and z-directions, giving total model dimensions of 124x128x124. At every time step, the variables calculated for each segment will be stored on the DataVault.

Figure 11: Segmentation of large model using DataVault

Appendices

A NRL cstar library

The NRL cstar library consists of functions written to provide a simple interface to many of the Connection Machine features that can only be accessed directly via C/Paris. The library modules were written to solve specific problems encountered by users at NRL. A full description of the library can be found in the **README** file located in the nrl-cstar-lib directory on the NRL and NPAC front-end computers (cmvax, cmsun and cmx).

The NRL C* library resides on the NRL front-end computers in the directory:

```
/cm-lights/library/nrl-cstar-lib
```

On the NPAC front-end computer, the NRL C* library is located in the directory:

```
/public/cm-lights/nrl-cstar-lib
```

A.1 N-dimensional grid

The finite differences method uses many 'nearest-neighbor' calculations. Although C* supports a NEWS package to perform nearest neighbor calculations in two dimensions, this package currently does not support three-dimensional NEWS communications. The **nd_grid** module of the NRL C* library provides fast nearest neighbor communications on N-dimensional grids (up to 8 dimensions).

For more information on the NRL C* n-dimensional grid library(figure 12):

```
cmx% cd /public/cm-lights/nrl-cstar-lib/nd_grid
cmx% more README
```

A.2 Framebuffer

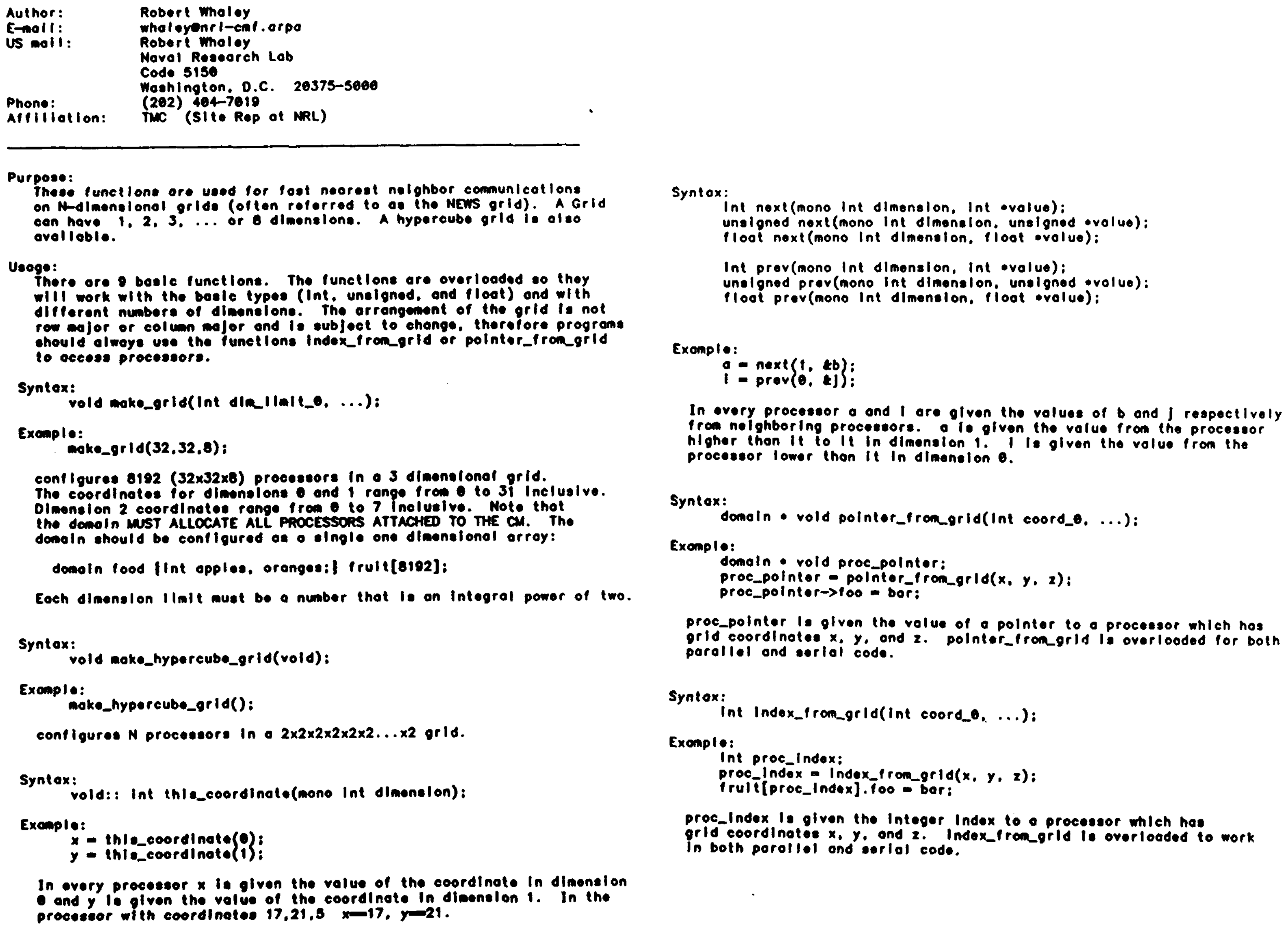
The NRL C* library module **fb** is a simple interface to the high speed, high resolution CM graphics display system. This module consists of functions to initialize the framebuffer, set up a color table, and display pixels on the color monitor.

For more information on the NRL C* framebuffer library (figure 13):

```
cmx% cd /public/cm-lights/nrl-cstar-lib/fb
cmx% more README
```

B cmusers group

NRL supports a users' forum for exchanging questions, ideas and experiences among programmers using the Connection Machine Facility. Mail can be directed to cmusers and questions usually



```
Author:         Robert Whaley
E-mail:         whaley@nrl-cmf.arpa
US mail:        Robert Whaley
                Naval Research Lab
                Code 5150
                Washington, D.C.  20375-5000
Phone:          (202) 404-7019
Affiliation:    TMC  (Site Rep at NRL)
_______________________________________________________________

Purpose:
   These functions are used for fast nearest neighbor communications
   on N-dimensional grids (often referred to as the NEWS grid).  A Grid
   can have  1, 2, 3, ... or 8 dimensions.  A hypercube grid is also
   available.

Usage:
   There are 9 basic functions.  The functions are overloaded so they
   will work with the basic types (int, unsigned, and float) and with
   different numbers of dimensions.  The arrangement of the grid is not
   row major or column major and is subject to change, therefore programs
   should always use the functions index_from_grid or pointer_from_grid
   to access processors.

 Syntax:
       void make_grid(int dim_limit_0, ...);

 Example:
       make_grid(32,32,8);

   configures 8192 (32x32x8) processors in a 3 dimensional grid.
   The coordinates for dimensions 0 and 1 range from 0 to 31 inclusive.
   Dimension 2 coordinates range from 0 to 7 inclusive.  Note that
   the domain MUST ALLOCATE ALL PROCESSORS ATTACHED TO THE CM.  The
   domain should be configured as a single one dimensional array:

     domain food {int apples, oranges;} fruit[8192];

   Each dimension limit must be a number that is an integral power of two.


 Syntax:
       void make_hypercube_grid(void);

 Example:
       make_hypercube_grid();

   configures N processors in a 2x2x2x2x2x2...x2 grid.


 Syntax:
       void:: int this_coordinate(mono int dimension);

 Example:
       x = this_coordinate(0);
       y = this_coordinate(1);

   In every processor x is given the value of the coordinate in dimension
   0 and y is given the value of the coordinate in dimension 1.  In the
   processor with coordinates 17,21,5  x=17, y=21.
```

```
Syntax:
      int next(mono int dimension, int *value);
      unsigned next(mono int dimension, unsigned *value);
      float next(mono int dimension, float *value);

      int prev(mono int dimension, int *value);
      unsigned prev(mono int dimension, unsigned *value);
      float prev(mono int dimension, float *value);


Example:
      a = next(1, &b);
      i = prev(0, &j);

  In every processor a and i are given the values of b and j respectively
  from neighboring processors.  a is given the value from the processor
  higher than it to it in dimension 1.  i is given the value from the
  processor lower than it in dimension 0.


Syntax:
      domain * void pointer_from_grid(int coord_0, ...);

Example:
      domain * void proc_pointer;
      proc_pointer = pointer_from_grid(x, y, z);
      proc_pointer->foo = bar;

  proc_pointer is given the value of a pointer to a processor which has
  grid coordinates x, y, and z.  pointer_from_grid is overloaded for both
  parallel and serial code.


Syntax:
      int index_from_grid(int coord_0, ...);

Example:
      int proc_index;
      proc_index = index_from_grid(x, y, z);
      fruit[proc_index].foo = bar;

  proc_index is given the integer index to a processor which has
  grid coordinates x, y, and z.  index_from_grid is overloaded to work
  in both parallel and serial code.
```

Figure 12: README file for nd_grid library

```
Author:         Robert Whaley
E-mail:         whaley@nrl-cmf.arpa
US mail:        Robert Whaley
                Naval Research Lab
                Code 5150
                Washington, D.C.  20375-5000
Phone;          (202) 404-7019
Affiliate:      TMC  (Site Rep at NRL)
_____________________________________________________

Purpose:
   These functions are used to display pixels on the framebuffer.

Usage:
   The framebuffer is first initialized, then the color map can be set,
   pixels displayed and finally the framebuffer released.

 Syntax:
      void init_frame_buffer(int x_size, int y_size);

 Example:
      init_frame_buffer(512, 512);

   Attach the framebuffer and configure it so that there are 512 pixels
   horizontally and 512 pixels vertically.


 Syntax:
      set_color(int color_id, int red, int green, int blue);

 Example:
      set_color(100, 128, 0, 128);

   Set the color map so that the color 100 is dark purple.  The default
   color map is gray scale from 0 (black), to 255 (white).  255 colors can
   be set.  Red, green, and blue can each range from 0 to 255 giving a total
   of 16 million possible shades.  Color 0 is the background color, so it is
   usually a good idea to leave it black (red=0, green=0, blue=0).

 Example:
      for (i=0; i<25; i++) set_color(i+200, i*10, 0, 250 - i*10);

   Set the color map values from 200 through 224 to range from bright
   blue (value 200) to bright red (value 224) with progressive shades
   of purple between.


 Syntax:
      plot_from_grid(unsigned char color);

 Example:
      unsigned char heat;
      plot_from_grid(heat);

   Each selected processor updates a single pixel to the color specified by
   the variable 'heat'.  The pixels updated are the ones with the same x, y
   coordinates as the processor.  It is an error to call this routine if
   the processors are not arranged as a 2-d grid.


 Syntax:
      plot_x_y(unsigned short x, unsigned short y, unsigned char color);

 Example:
      unsigned short pix_x, pix_y;
      unsigned char pix_color;
      plot_x_y(pix_x, pix_y, pix_color);

   Each selected processor sets the pixel at 'pix_x', 'pix_y' to the
   color 'pix_color'.


 Syntax:
      void release_frame_buffer(void);

 Example:
      release_frame_buffer();

   Frees the framebuffer to be used by other programs.
```

Figure 13: README file for fb library

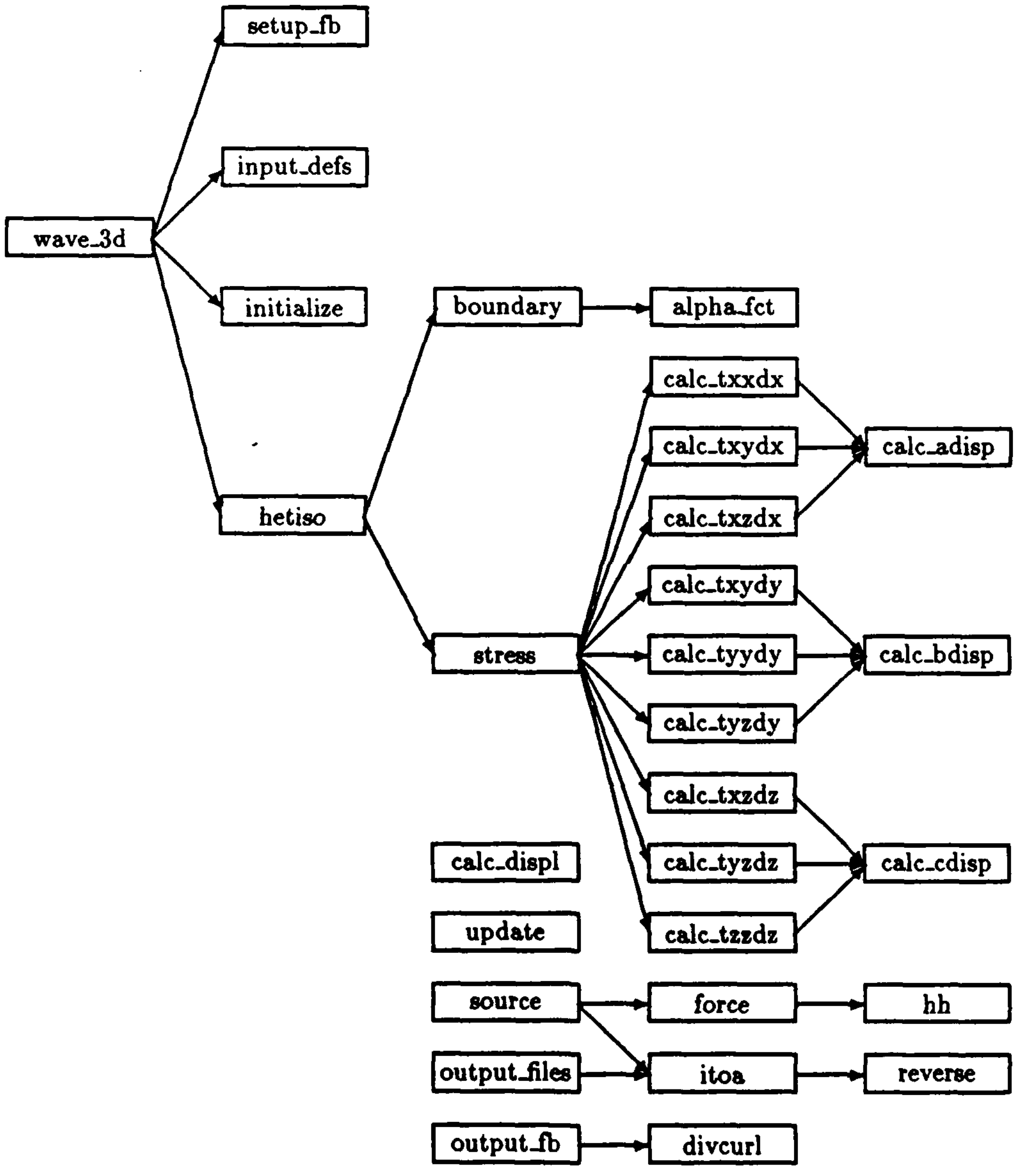
receive quick and thorough responses. Since all members of the cmusers group receive the mail, you can learn about problems/bugs with new compilers, upcoming classes and talks, innovative algorithms, etc. To join this forum, simply mail your request to be added to the mailing list to *cmusers*.

To get help on the mail facility:

```
cmsun% man mail
```

NPAC supports an online consultant service. To get help, send mail to *consult*.

C Structure diagram for wave_3d.cs



D Source Code

D.1 wave_3d program

```
/* ========================================================= */
/*      program wave_3d                                      */
/*                                                           */
/*      Programmer:  Julie Allen                             */
/*      Date:        28 February 1989                        */
/*                                                           */
/* ========================================================= */
/*                                                           */
/*        Copyright (c) 1989                                 */
/*        Information Processing and Communications Laboratory */
/*        Woods Hole Oceanographic Institution               */
/*        All rights reserved.                               */
/*                                                           */
/*        This material cannot be distributed or sold without */
/*                prior permission of the author(s).        */
/*                                                           */
/* ========================================================= */
/*                                                           */
/*      %cs -O -o wave_3d wave_3d.cs nd_grid.o               */
/*      %cmattach -b 0.95 -p 16k -v 256k -C S wave_3d &      */
/*                                                           */
/* ========================================================= */
#include        <stdio.hs>
#include        <cm/cmtimer.hs>
#include        <math.hs>
#include        <string.h>
#include        <ctype.h>
#include        "nd_grid.hs"
#include        "fb.hs"
#include        "wave_3d.hs"

void main()
{
    int i;

    /* function declarations */
    void input_defs(void);
    void hetiso(void);
    void make_grid(int d0, int d1, int d2);
    void setup_fb(void);

    printf("\n     ********** Program wave_3d **********\n\n");
    printf("3D finite difference modeling program\n");
    printf("Running on Thinking Machines CM/2 Parallel Processor\n");
    printf("Array size: %d  XDIM: %d  YDIM: %d  ZDIM: %d\n",
        TOTAL, XDIM, YDIM, ZDIM);

    /* define the 3D grid */
    [domain model].{ make_grid(XDIM,YDIM,ZDIM);}

    /* setup the frame buffer and color table */
    setup_fb();

    /* input model definition and parameters */
    TIMER_ON;                   /* start timer */
    (void) input_defs();

    printf("Time for function input_defs:\n");
    TIMER_OFF;                  /* stop timer */
    TIMER_ON;                   /* start timer */

    /* set up initialize conditions */
    [domain model].{initialize();}
    printf("Time for initialization:\n");
    TIMER_OFF;                  /* stop timer */
    TIMER_ON;                   /* start timer */

    /* perform hetergeneous/isotropic finite difference calculations */
    (void) hetiso();

    printf("Time for rest of program:\n");
    TIMER_OFF;                  /* stop timer */
    printf("End of program wave_3d.cs\n");
}
/*============================================================== */
/*                                                              */
/*      Function alpha_fct                                      */
/*                                                              */
/*============================================================== */
float model::alpha_fct(int indx)
{
    if(indx < NTEL)
        return(sin(((NTEL-indx-1)/(NTEL/2))*PI/2.f)*(0.10f));
    else
        return(sin(((indx-(ZDIM-NTEL))/(NTEL/2))*PI/2.f)*(0.10f));
}

/* ============================================================= */
/*                                                              */
/*      Function boundary                                       */
/*                                                              */
/* ============================================================== */
void model::boundary(void)
{
    int ix,iy,iz;
    float alphx = 0.0f, alphy = 0.0f, alphz = 0.0f;

    alpha = 0.0f;

    ix = this_coordinate(0);
    iy = this_coordinate(1);
    iz = this_coordinate(2);

    if(iz < NTEL || iz >= (ZDIM - NTEL))
        if(iz < NTEL/2 || iz >= (ZDIM - NTEL/2))
            alphz = 0.10f;
        else
            alphz = alpha_fct(iz);

    if(iy < NTEL || iy >= (YDIM - NTEL))
        if(iy < NTEL/2 || iy >= (YDIM - NTEL/2))
            alphy = 0.10f;
        else
            alphy = alpha_fct(iy);

    if(ix < NTEL || ix >= (XDIM - NTEL))
        if(ix < NTEL/2 || ix >= (XDIM - NTEL/2))
            alphx = 0.10f;
        else
            alphx = alpha_fct(ix);

    alpha = (alphx > alphy) ? alphx : alphy;
    alpha = (alphz > alpha) ? alphz : alpha;
}
/* ============================================================== */
/*                                                                */
/*       Function calc_adisp                                      */
/*                                                                */
/* ============================================================== */
void model::calc_adisp(float *adisp)
{
    float tn0u,tn1u,tn2u,tn0v,tn0w;
    float tp0u,tp0v,tp1v,tp0w;

    /* set up temporary 'next' values */
    tn0u = next(0,&u1);
    tn1u = next(1,&u1);
    tn2u = next(2,&u1);
    tn0v = next(0,&v1);
    tn0w = next(0,&w1);

    /* set up temporary 'prev' values */
    tp0u = prev(0,&u1);
    tp0v = prev(0,&v1);
    tp1v = prev(1,&v1);
    tp0w = prev(0,&w1);

    /* term 1: differences placed in 'a' array
               consists of difference terms used in the d/dx stress term */

    *(adisp+0) = (tn0u - u1) * xlamb;
    *(adisp+1) = (u1 - tp0u) * xlamb;
    *(adisp+2) = (v1 - tp1v) * xylamb;
    *(adisp+3) = (tp0v - prev(1,&tp0v)) * xylamb;
    *(adisp+4) = (w1 - prev(2,&w1)) * xzlamb;
    *(adisp+5) = (tp0w - prev(2,&tp0w)) * xzlamb;
    *(adisp+6) = (next(2,&tn0u) - tn0u) * xzlamb;
    *(adisp+7) = (tn2u - u1) * xzlamb;
    *(adisp+8) = (tn0w - w1) * xlamb;
    *(adisp+9) = (w1 - tp0w) * xlamb;
    *(adisp+10) = (next(1,&tn0u) - tn0u) * xylamb;
    *(adisp+11) = (tn1u - u1) * xylamb;
    *(adisp+12) = (tn0v - v1) * xlamb;
    *(adisp+13) = (v1 - tp0v) * xlamb;
}

/* ============================================================== */
/*                                                                */
/*       Function calc_bdisp                                      */
/*                                                                */
/* ============================================================== */
void model::calc_bdisp(float *bdisp)
{
    float tn0u,tn1u,tn1v,tn2v,tn1w;
    float tp1u,tp0v,tp1v,tp1w,tp2w;

    /* set up temporary 'next' values */
    tn0u = next(0,&u1);
    tn1u = next(1,&u1);
    tn1v = next(1,&v1);
    tn2v = next(2,&v1);
    tn1w = next(1,&w1);

    /* set up temporary 'prev' values */
    tp1u = prev(1,&u1);
    tp0v = prev(0,&v1);
    tp1v = prev(1,&v1);
    tp1w = prev(1,&w1);
    tp2w = prev(2,&w1);

    /* term 2: differences placed in 'b' array
               terms used in the d/dy stress term  */
    *(bdisp+0) = (next(1,&tn0u) - tn1u) * xylamb;
    *(bdisp+1) = (tn0u - u1) * xylamb;
    *(bdisp+2) = (tn1v - v1) * ylamb;
    *(bdisp+3) = (v1 - tp1v) * ylamb;
    *(bdisp+4) = (tn1w - prev(2,&tn1w)) * yzlamb;
    *(bdisp+5) = (w1 - tp2w) * yzlamb;
    *(bdisp+6) = (tn1u - u1) * ylamb;
    *(bdisp+7) = (u1 - tp1u) * ylamb;
    *(bdisp+8) = (v1 - tp0v) * xylamb;
    *(bdisp+9) = (tp1v - prev(1,&tp0v)) * xylamb;
    *(bdisp+10) = (tn2v - v1) * yzlamb;
    *(bdisp+11) = (next(2,&tp1v) - tp1v) * yzlamb;
    *(bdisp+12) = (tn1w - w1) * ylamb;
    *(bdisp+13) = (w1 - tp1w) * ylamb;
}
/* ============================================================ */
/*                                                              */
/*       Function calc_cdisp                                    */
/*                                                              */
/* ============================================================ */
void model::calc_cdisp(float *cdisp)
{
    float tn0u,tn2u,tn2v,tn1w,tn2w;
    float tp2u,tp1v,tp2v,tp0w,tp2w;

    /* set up temporary 'next' values */
    tn0u = next(0,&u1);
    tn2u = next(2,&u1);
    tn2v = next(2,&v1);
    tn1w = next(1,&w1);
    tn2w = next(2,&w1);

    /* set up temporary 'prev' values */
    tp2u = prev(2,&u1);
    tp1v = prev(1,&v1);
    tp2v = prev(2,&v1);
    tp0w = prev(0,&w1);
    tp2w = prev(2,&w1);

    /* term 3: differences placed in 'c' array
           consists of difference terms used in the d/dz stress term */
    *(cdisp+0) = (next(2,&tn0u) - tn2u) * xzlamb;
    *(cdisp+1) = (tn0u - u1) * xzlamb;
    *(cdisp+2) = (tn2v - next(2,&tp1v)) * yzlamb;
    *(cdisp+3) = (v1 - tp1v) * yzlamb;
    *(cdisp+4) = (tn2w - w1) * zlamb;
    *(cdisp+5) = (w1 - tp2w) * zlamb;
    *(cdisp+6) = (tn2u - u1) * zlamb;
    *(cdisp+7) = (u1 - tp2u) * zlamb;
    *(cdisp+8) = (w1 - tp0w) * xzlamb;
    *(cdisp+9) = (tp2w - prev(2,&tp0w)) * xzlamb;
    *(cdisp+10) = (tn2v - v1) * zlamb;
    *(cdisp+11) = (v1 - tp2v) * zlamb;
    *(cdisp+12) = (tn1w - w1) * yzlamb;
    *(cdisp+13) = (prev(2,&tn1w) - tp2w) * yzlamb;
}


/* ============================================================== */
/*                                                                */
/*          Function calc_displ                                   */
/*                                                                */
/* ============================================================== */
void model::calc_displ(void)
{
    int ix,iy,iz;
    float alph1,alph2,alph3;

    ix = this_coordinate(0);
    iy = this_coordinate(1);
    iz = this_coordinate(2);

    if((ix < 1 || ix >= XDIM) || (iy < 1 || iy >= YDIM) ||
        (iz < 1 || iz >= ZDIM))
        ;
    else
    {
        alph1 = 2.f - alpha * alpha;
        alph2 = 1.f - 2.f * alpha;
        alph3 = 1.f + 2.f * alpha;

        u0 = (alph1 * u1 - alph2 * u2 + txxdx + txydy + txzdz)/alph3;
        v0 = (alph1 * v1 - alph2 * v2 + txydx + tyydy + tyzdz)/alph3;
        w0 = (alph1 * w1 - alph2 * w2 + txzdx + tyzdy + tzzdz)/alph3;
    }
}
/* ============================================================== */
/*                                                                */
/*          Function calc_txxdx                                   */
/*                                                                */
/* ============================================================== */
```

```
void model::calc_txxdx(void)
{
    int ix,iy,iz;
    float alp2u1,alp2u2;
    float amuxz1,amuxz2,amuxz3,amuxz4,amuxz5,amuxz6;
    float amuxy1,amuxy2,amuxy3,amuxy4,amuxy5,amuxy6;
    float amuyz1,amuyz2,amuyz3,amuyz4,amuyz5,amuyz6;
    float amuyz7,amuyz8;
    float abigu1,abigu2;
    float alamb1,alamb2;
    float temp;
    float tn0,tn1,tn2,tp0,tp1,tp2;
    float adisp[14];

    /* first get the displacement difference terms */
    calc_adisp(adisp);

    /* set up temporary values */
    tn0 = next(0,&clp2mu);
    tp0 = prev(0,&clp2mu);

    /* find the lambda+2mu values for the txx,tyy,tzz positions */
    alp2u1 = (clp2mu + tn0)/2.f;
    alp2u2 = (clp2mu + tp0)/2.f;

    /* set up temporary values */
    tn0 = next(0,&cmu);
    tn1 = next(1,&cmu);
    tn2 = next(2,&cmu);
    tp0 = prev(0,&cmu);
    tp1 = prev(1,&cmu);
    tp2 = prev(2,&cmu);

    /* next, the mu value for the txz positions */
    amuxz1 = (cmu + tn2)/2.f;
    amuxz2 = (next(0,&cmu) + next(2,&tn0))/2.f;
    amuxz3 = (cmu + prev(2,&cmu))/2.f;
    amuxz4 = (next(0,&cmu) + prev(2,&tn0))/2.f;
    amuxz5 = (prev(0,&cmu) + next(2,&tp0))/2.f;
    amuxz6 = (prev(0,&cmu) + prev(2,&tp0))/2.f;

    /* now, the mu for the txy positions */
    amuxy1 = (cmu + next(1,&cmu))/2.f;
    amuxy2 = (next(0,&cmu) + next(1,&tn0))/2.f;
    amuxy3 = (cmu + prev(1,&cmu))/2.f;
    amuxy4 = (next(0,&cmu) + prev(1,&tn0))/2.f;
    amuxy5 = (prev(0,&cmu) + next(1,&tp0))/2.f;
    amuxy6 = (prev(0,&cmu) + prev(1,&tp0))/2.f;
    /* finally, the mu for the tyz positions */

    temp = next(1,&tn0);
    amuyz1 = (cmu + tn0 + tn1 + next(1,&tn0) + tn2 +
              next(2,&tn0) + next(2,&tn1) + next(2,&temp))/8.f;
    temp = next(1,&tn0);
    amuyz2 = (cmu + tn0 + tn1 + next(1,&tn0) + tp2 +
              prev(2,&tn0) + prev(2,&tn1) + prev(2,&temp))/8.f;
    temp = prev(1,&tn0);
    amuyz3 = (cmu + tn0 + tp1 + prev(1,&tn0) + tn2 +
              next(2,&tn0) + next(2,&tp1) + next(2,&temp))/8.f;
    temp = prev(1,&tn0);
    amuyz4 = (cmu + tn0 + tp1 + prev(1,&tn0) + tp2 +
              prev(2,&tn0) + prev(2,&tp1) + prev(2,&temp))/8.f;
    temp = next(1,&tp0);
    amuyz5 = (cmu + tp0 + tn1 + next(1,&tp0) + tn2 +
              next(2,&tp0) + next(2,&tn1) + next(2,&temp))/8.f;
    temp = next(1,&tp0);
    amuyz6 = (cmu + tp0 + tn1 + next(1,&tp0) + tp2 +
              prev(2,&tp0) + prev(2,&tn1) + prev(2,&temp))/8.f;
    temp = prev(1,&tp0);
    amuyz7 = (cmu + tp0 + tp1 + prev(1,&tp0) + tn2 +
              next(2,&tp0) + next(2,&tp1) + next(2,&temp))/8.f;
    temp = prev(1,&tp0);
    amuyz8 = (cmu + tp0 + tp1 + prev(1,&tp0) + tp2 +
              prev(2,&tp0) + prev(2,&tp1) + prev(2,&temp))/8.f;

    /* next, get the lambda value for txx,tyy,tzz positions */
    abigu1 = (amuxz1 + amuxz2 + amuxz3 + amuxz4 +
              amuxy1 + amuxy2 + amuxy3 + amuxy4 +
              amuyz1 + amuyz2 + amuyz3 + amuyz4)/12.f;
    abigu2 = (amuxz1 + amuxz3 + amuxz5 + amuxz6 +
              amuxy1 + amuxy3 + amuxy5 + amuxy6 +
              amuyz5 + amuyz6 + amuyz7 + amuyz8)/12.f;

    alamb1 = alp2u1 - 2.f * abigu1;
    alamb2 = alp2u2 - 2.f * abigu2;

    txxdx = (alp2u1 * adisp[0] - alp2u2 * adisp[1]) +
            (alamb1 * adisp[2] - alamb2 * adisp[3]) +
            (alamb1 * adisp[4] - alamb2 * adisp[5]);

    /* now include density term */
    txxdx = txxdx/rho;
}

/* ============================================================== */
/*                                                                */
/*        Function calc_txydx                                         */
/*                                                                    */
/* ================================================================== */
void model::calc_txydx(void)
{
    float amuxy1,amuxy2;
    float rhov;
    float tn0,tn1;
    float adisp[14];

    /* first get the displacement difference terms */
    calc_adisp(adisp);

    /* set up temporary values */
    tn0 = next(0,&cmu);
    tn1 = next(1,&cmu);

    /* now, the mu for the txy positions */
    amuxy1 = (cmu + next(1,&cmu))/2.f;
    amuxy2 = (next(0,&cmu) + next(1,&tn0))/2.f;

    txydx = (amuxy2 * adisp[10] - amuxy1 * adisp[11]) +
            (amuxy2 * adisp[12] - amuxy1 * adisp[13]);

    /* set up temporary values */
    tn0 = next(0,&rho);
    tn1 = next(1,&rho);

    /* density term for y-displacement */
    rhov = (rho + tn0 + tn1 + next(1,&tn0))/4.f;

    /* now include density term */
    txydx = txydx/rhov;
}

/* ================================================================== */
/*                                                                    */
/*        Function calc_txydy                                         */
/*                                                                    */
/* ================================================================== */
void model::calc_txydy(void)
{
    float amuxy1,amuxy3;
    float bdisp[14];

    /* first get the displacement difference terms */
    calc_bdisp(bdisp);
    /* now, the mu for the txy positions */
    amuxy1 = (cmu + next(1,&cmu))/2.f;
    amuxy3 = (cmu + prev(1,&cmu))/2.f;

    txydy = (amuxy1 * bdisp[6] - amuxy3 * bdisp[7]) +
            (amuxy1 * bdisp[8] - amuxy3 * bdisp[9]) ;

    /* now include density term */
    txydy = txydy/rho;
}

/* ============================================================== */
/*                                                                */
/*        Function calc_txzdx                                     */
/*                                                                */
/* ============================================================== */
void model::calc_txzdx(void)
{
    float amuxz1,amuxz2;
    float rhow;
    float tn0,tn2;
    float adisp[14];

    /* first get the displacement difference terms */
    calc_adisp(adisp);

    /* set up temporary values */
    tn0 = next(0,&cmu);
    tn2 = next(2,&cmu);

    /* next, the mu value for the txz positions */
    amuxz1 = (cmu + tn2)/2.f;
    amuxz2 = (next(0,&cmu) + next(2,&tn0))/2.f;

    txzdx = (amuxz2 * adisp[6] - amuxz1 * adisp[7]) +
            (amuxz2 * adisp[8] - amuxz1 * adisp[9]);

    /* set up temporary values */
    tn0 = next(0,&rho);
    tn2 = next(2,&rho);

    /* density term for z-displacement */
    rhow = (rho + tn0 + tn2 + next(2,&tn0))/4.f;

    /* now include density term */
    txzdx = txzdx/rhow;
}
/* ============================================================== */
/*                                                              */
/*        Function calc_txzdz                                   */
/*                                                              */
/* ============================================================ */
void model::calc_txzdz(void)
{
    float amuxz1,amuxz3;
    float tn2;
    float cdisp[14];

    /* first get the displacement difference terms */
    calc_cdisp(cdisp);

    /* set up temporary values */
    tn2 = next(2,&cmu);

    /* next, the mu value for the txz positions */
    amuxz1 = (cmu + tn2)/2.f;
    amuxz3 = (cmu + prev(2,&cmu))/2.f;

    txzdz = (amuxz1 * cdisp[6] - amuxz3 * cdisp[7]) +
            (amuxz1 * cdisp[8] - amuxz3 * cdisp[9]);

    /* now include density term */
    txzdz = txzdz/rho;
}
/* ============================================================ */
/*                                                              */
/*         Function calc_tyydy                                  */
/*                                                              */
/* ============================================================ */
void model::calc_tyydy(void)
{
    float alp2u1,alp2u3;
    float amuxz1,amuxz2,amuxz3,amuxz4,
          amuxz7,amuxz8,amuxz9,amuxza;
    float amuxy1,amuxy2,amuxy3,amuxy4,amuxy7,amuxy8;
    float amuyz1,amuyz2,amuyz3,amuyz4,amuyz9,amuyza;
    float abigu1,abigu3;
    float alamb1,alamb3;
    float rhov;
    float temp,temp2,temp3,temp4;
    float tn0,tn1,tn2,tp0,tp1,tp2;
    float bdisp[14];

    /* first get the displacement difference terms */
    calc_bdisp(bdisp);
```

```
/* set up temporary values */
tn0 = next(0,&clp2mu);

/* find the lambda+2mu values for the txx,tyy,tzz positions */
alp2u1 = (clp2mu + tn0)/2.f;
alp2u3 = (next(1,&clp2mu) + next(1,&tn0))/2.f;

/* set up temporary values */
tn0 = next(0,&cmu);
tn1 = next(1,&cmu);
tn2 = next(2,&cmu);
tp0 = prev(0,&cmu);
tp1 = prev(1,&cmu);
tp2 = prev(2,&cmu);

/* next, the mu value for the txz positions */
amuxz1 = (cmu + tn2)/2.f;
amuxz2 = (next(0,&cmu) + next(2,&tn0))/2.f;
amuxz3 = (cmu + prev(2,&cmu))/2.f;
amuxz4 = (next(0,&cmu) + prev(2,&tn0))/2.f;
amuxz7 = (next(1,&cmu) + next(2,&tn1))/2.f;
temp = next(1,&tn0);
amuxz8 = (next(1,&tn0) + next(2,&temp))/2.f;
amuxz9 = (next(1,&cmu) + prev(2,&tn1))/2.f;
temp = next(1,&tn0);
amuxza = (next(1,&tn0) + prev(2,&temp))/2.f;
/* now, the mu for the txy positions */
amuxy1 = (cmu + next(1,&cmu))/2.f;
amuxy2 = (next(0,&cmu) + next(1,&tn0))/2.f;
amuxy3 = (cmu + prev(1,&cmu))/2.f;
amuxy4 = (next(0,&cmu) + prev(1,&tn0))/2.f;
amuxy7 = (next(1,&cmu) + next(1,&tn1))/2.f;
temp = next(1,&tn0);
amuxy8 = (next(1,&tn0) + next(1,&temp))/2.f;

/* finally, the mu for the tyz positions */
temp = next(1,&tn0);
amuyz1 = (cmu + tn0 + tn1 + next(1,&tn0) + tn2 +
          next(2,&tn0) + next(2,&tn1) + next(2,&temp))/8.f;
temp = next(1,&tn0);
amuyz2 = (cmu + tn0 + tn1 + next(1,&tn0) + tp2 +
          prev(2,&tn0) + prev(2,&tn1) + prev(2,&temp))/8.f;
temp = prev(1,&tn0);
amuyz3 = (cmu + tn0 + tp1 + prev(1,&tn0) + tn2 +
          next(2,&tn0) + next(2,&tp1) + next(2,&temp))/8.f;
temp = prev(1,&tn0);
amuyz4 = (cmu + tn0 + tp1 + prev(1,&tn0) + tp2 +
          prev(2,&tn0) + prev(2,&tp1) + prev(2,&temp))/8.f;
```

```
    temp = next(1,&tn0);
    temp2 = next(1,&tn2);
    temp3 = next(1,&tn0);
    temp3 = next(1,&temp3);
    temp4 = next(1,&tn0);
    amuyz9 = (cmu + next(1,&temp) + tn1 + temp + next(1,&temp2) +
              next(2,&temp3) + next(2,&tn1) + next(2,&temp4))/8.f;
    temp = next(1,&tn0);
    temp2 = next(1,&tn1);
    temp3 = next(1,&tn0);
    temp3 = next(1,&temp3);
    temp4 = next(1,&tn0);
    amuyza = (next(1,&tn1) + next(1,&temp) + tn1 + next(1,&tn0) +
              prev(2,&temp2) + prev(2,&temp3) + prev(2,&tn1) +
              prev(2,&temp4))/8.f;

    /* next, get the lambda value for txx,tyy,tzz positions */
    abigu1 = (amuxz1 + amuxz2 + amuxz3 + amuxz4 +
              amuxy1 + amuxy2 + amuxy3 + amuxy4 +
              amuyz1 + amuyz2 + amuyz3 + amuyz4)/12.f;
    abigu3 = (amuxz7 + amuxz8 + amuxz9 + amuxza +
              amuxy7 + amuxy8 + amuxy1 + amuxy2 +
              amuyz1 + amuyz2 + amuyz9 + amuyza)/12.f;

    alamb1 = alp2u1 - 2.f * abigu1;
    alamb3 = alp2u3 - 2.f * abigu3;

    tyydy = (alamb3 * bdisp[0] - alamb1 * bdisp[1]) +
            (alp2u3 * bdisp[2] - alp2u1 * bdisp[3]) +
            (alamb3 * bdisp[4] - alamb1 * bdisp[5]);

    /* set up temporary values */
    tn0 = next(0,&rho);
    tn1 = next(1,&rho);

    /* density term for y-displacement */
    rhov = (rho + tn0 + tn1 + next(1,&tn0))/4.f;

    /* now include density term */
    tyydy = tyydy/rhov;
}

/* ================================================================ */
/*                                                                  */
/*         Function calc_tyzdy                                      */
/*                                                                  */
/* ================================================================ */
void model::calc_tyzdy(void)
```

```
{
    float amuyz1,amuyz3;
    float rhow;
    float temp;
    float tn0,tn1,tn2,tp1;
    float bdisp[14];

    /* first get the displacement difference terms */
    calc_bdisp(bdisp);

    /* set up temporary values */
    tn0 = next(0,&cmu);
    tn1 = next(1,&cmu);
    tn2 = next(2,&cmu);
    tp1 = prev(1,&cmu);

    /* finally, the mu for the tyz positions */
    temp = next(1,&tn0);
    amuyz1 = (cmu + tn0 + tn1 + next(1,&tn0) + tn2 +
              next(2,&tn0) + next(2,&tn1) + next(2,&temp))/8.f;
    temp = prev(1,&tn0);
    amuyz3 = (cmu + tn0 + tp1 + prev(1,&tn0) + tn2 +
    next(2,&tn0) + next(2,&tp1) + next(2,&temp))/8.f;

    tyzdy = (amuyz1 * bdisp[10] - amuyz3 * bdisp[11]) +
            (amuyz1 * bdisp[12] - amuyz3 * bdisp[13]);

    /* set up temporary values */
    tn0 = next(0,&rho);
    tn2 = next(2,&rho);

    /* density term for z-displacement */
    rhow = (rho + tn0 + tn2 + next(2,&tn0))/4.f;

    /* now include density term */
    tyzdy = tyzdy/rhow;
}
/* ================================================================= */
/*                                                                   */
/*          Function calc_tyzdz                                      */
/*                                                                   */
/* ================================================================= */
void model::calc_tyzdz(void)
{
    float amuyz1,amuyz2;
    float rhov;
    float temp;
    float tn0,tn1,tn2,tp2;
    float cdisp[14];

    /* first get the displacement difference terms */
    calc_cdisp(cdisp);

    /* set up temporary values */
    tn0 = next(0,&cmu);
    tn1 = next(1,&cmu);
    tn2 = next(2,&cmu);
    tp2 = prev(2,&cmu);

    /* finally, the mu for the tyz positions */
    temp = next(1,&tn0);
    amuyz1 = (cmu + tn0 + tn1 + next(1,&tn0) + tn2 +
              next(2,&tn0) + next(2,&tn1) + next(2,&temp))/8.f;
    temp = next(1,&tn0);
    amuyz2 = (cmu + tn0 + tn1 + next(1,&tn0) + tp2 +
              prev(2,&tn0) + prev(2,&tn1) + prev(2,&temp))/8.f;

    tyzdz = (amuyz1 * cdisp[10] - amuyz2 * cdisp[11]) +
            (amuyz1 * cdisp[12] - amuyz2 * cdisp[13]);
    /* set up temporary values */
    tn0 = next(0,&rho);
    tn1 = next(1,&rho);

    /* density term for y-displacement */
    rhov = (rho + tn0 + tn1 + next(1,&tn0))/4.f;

    /* now include density term */
    tyzdz = tyzdz/rhov;
}
/* ============================================================ */
/*                                                              */
/*          Function calc_tzzdz                                 */
/*                                                              */
/* ============================================================ */
void model::calc_tzzdz(void)
{
    float alp2u1,alp2u4;
    float amuxz1,amuxz2,amuxz3,amuxz4,amuxzb,amuxzc;
    float amuxy1,amuxy2,amuxy3,amuxy4,
          amuxy9,amuxya,amuxyb,amuxyc;
    float amuyz1,amuyz2,amuyz3,amuyz4,amuyzb,amuyzc;
    float abigu1,abigu4;
    float alamb1,alamb4;
    float rhow;
    float temp,temp2,temp3,temp4;
    float tn0,tn1,tn2,tp1,tp2;
float cdisp[14];

/* first get the displacement difference terms */
calc_cdisp(cdisp);

/* set up temporary values */
tn0 = next(0,&clp2mu);

/* find the lambda+2mu values for the txx,tyy,tzz positions */
alp2u1 = (clp2mu + tn0)/2.f;
alp2u4 = (next(2,&clp2mu) + next(2,&tn0))/2.f;

/* set up temporary values */
tn0 = next(0,&cmu);
tn1 = next(1,&cmu);
tn2 = next(2,&cmu);
tp1 = prev(1,&cmu);
tp2 = prev(2,&cmu);

/* next, the mu value for the txz positions */
amuxz1 = (cmu + tn2)/2.f;
amuxz2 = (next(0,&cmu) + next(2,&tn0))/2.f;
amuxz3 = (cmu + prev(2,&cmu))/2.f;
amuxz4 = (next(0,&cmu) + prev(2,&tn0))/2.f;
amuxzb = (next(2,&cmu) + next(2,&tn2))/2.f;
temp2 = next(2,&tn0);
amuxzc = (next(2,&tn0) + next(2,&temp))/2.f;

/* now, the mu for the txy positions */
amuxy1 = (cmu + next(1,&cmu))/2.f;
amuxy2 = (next(0,&cmu) + next(1,&tn0))/2.f;
amuxy3 = (cmu + prev(1,&cmu))/2.f;
amuxy4 = (next(0,&cmu) + prev(1,&tn0))/2.f;
amuxy9 = (next(2,&cmu) + next(2,&tn1))/2.f;
temp = next(1,&tn0);
amuxya = (next(2,&tn0) + next(2,&temp))/2.f;
amuxyb = (next(2,&cmu) + next(2,&tp1))/2.f;
temp = prev(1,&tn0);
amuxyc = (next(2,&tn0) + next(2,&temp))/2.f;
/* finally, the mu for the tyz positions */
temp = next(1,&tn0);
amuyz1 = (cmu + tn0 + tn1 + next(1,&tn0) + tn2 +
          next(2,&tn0) + next(2,&tn1) + next(2,&temp))/8.f;
temp = next(1,&tn0);
amuyz2 = (cmu + tn0 + tn1 + next(1,&tn0) + tp2 +
          prev(2,&tn0) + prev(2,&tn1) + prev(2,&temp))/8.f;
temp = prev(1,&tn0);
amuyz3 = (cmu + tn0 + tp1 + prev(1,&tn0) + tn2 +
    next(2,&tn0) + next(2,&tp1) + next(2,&temp))/8.f;
    temp = prev(1,&tn0);
    amuyz4 = (cmu + tn0 + tp1 + prev(1,&tn0) + tp2 +
              prev(2,&tn0) + prev(2,&tp1) + prev(2,&temp))/8.f;
    temp = next(2,&tn0);
    temp2 = next(2,&tn1);
    temp3 = next(1,&tn0);
    temp3 = next(2,&temp3);
    temp4 = next(1,&tn0);
    amuyzb = (next(2,&tn2) + next(2,&temp) + next(2,&temp2) +
              next(2,&temp3) + tn2 + next(2,&tn0) + next(2,&tn1) +
              next(2,&temp4))/8.f;
    temp = next(2,&tn0);
    temp2 = next(2,&tp1);
    temp3 = prev(1,&tn0);
    temp3 = next(2,&temp3);
    temp4 = prev(1,&tn0);
    amuyzc = (next(2,&tn2) + next(2,&temp) + next(2,&temp2) +
              next(2,&temp3) + tn2 + next(2,&tn0) + next(2,&tp1) +
              next(2,&temp4))/8.f;

    /* next, get the lambda value for txx,tyy,tzz positions */
    abigu1 = (amuxz1 + amuxz2 + amuxz3 + amuxz4 +
              amuxy1 + amuxy2 + amuxy3 + amuxy4 +
              amuyz1 + amuyz2 + amuyz3 + amuyz4)/12.f;
    abigu4 = (amuxz1 + amuxz2 + amuxzb + amuxzc +
              amuxy9 + amuxya + amuxyb + amuxyc +
              amuyz1 + amuyz3 + amuyzb + amuyzc)/12.f;

    alamb1 = alp2u1 - 2.f * abigu1;
    alamb4 = alp2u4 - 2.f * abigu4;

    tzzdz = (alamb4 * cdisp[0] - alamb1 * cdisp[1]) +
            (alamb4 * cdisp[2] - alamb1 * cdisp[3]) +
            (alp2u4 * cdisp[4] - alp2u1 * cdisp[5]) ;

    /* set up temporary values */
    tn0 = next(0,&rho);
    tn2 = next(2,&rho);

    /* density term for z-displacement */
    rhow = (rho + tn0 + tn2 + next(2,&tn0))/4.f;

    /* now include density term */
    tzzdz = tzzdz/rhow;
}

/* ================================================================ */
/*                                                              */
/*         Function divcurl - calculate the divergence and curl */
/*                                                              */
/* ============================================================ */
void model::divcurl(void)
{
    int ix,iy,iz;
    mono float div_min,div_max;

    /* zero out div and curl values */
    div = curlx = curly = curlz = 0.0;

    /* calculate divergence and curl */
    ix = this_coordinate(0);
    iy = this_coordinate(1);
    iz = this_coordinate(2);

    if((ix < 1 || ix >= XDIM) || (iy < 1 || iy >= YDIM) ||
        (iz < 1 || iz >= ZDIM))
        ;
    else
    {
        div = (next(0,&u1) - u1)/dx + (v1 - prev(1,&v1))/dy +
              (w1 - prev(2,&w1))/dz;
        curlx = (next(1,&w1) - w1)/dy - (next(2,&v1) - v1)/dz;
        curly = (next(2,&u1) - u1)/dz - (w1 - prev(0,&w1))/dx;
        curlz = (v1 - prev(0,&v1))/dx - (next(1,&u1) - u1)/dy;
    }

    /* get minimum and maximum div */
    div_min <?= div;
    div_max >?= div;
    printf("minimum div: %14.7e  maximum div: %14.7e\n", div_min,div_max);
}
/* ============================================================ */
/*                                                              */
/*         Function force                                       */
/*                                                              */
/* ============================================================ */
force(int ix,int jy,int kz,int ltime,float *fsorsu,float *fsorsv,float *fsorsw)
{
    int del;
    float ru,rv,rw,hx,hy,hz;
    float xpwidt,gtp,gt,dircsx,dircsy,dircsz;
    float time = ltime * dt;
    float hh(float r);

    del = 6 * dx;
    /* calculate range values (radii) for u,v, and w grid points */
    ru = sqrt((ix*ix) * (dx*dx) + (jy*jy) * (dy*dy) + (kz*kz) * (dz*dz));
    rv = sqrt(((ix+0.5f)*(ix+0.5f)) * (dx*dx) + ((jy+0.5f)*(jy+0.5f)) *
               (dy*dy) + (kz*kz) * (dz*dz));
    rw = sqrt(((ix+0.5f)*(ix+0.5f)) * (dx*dx) + (jy*jy) * (dy*dy) +
              ((kz+0.5f)*(kz+0.5f)) * (dz*dz));

    /* calculate 'jagged' sine wave for x,y, and z - spatial scaling function
       applied to the time dependent source function */
    hx = 2.f/(del*del) * (-ru*hh(ru) + 2.f * (ru - del/2.f)*hh(ru-del/2.f) -
             (ru-del)*hh(ru-del));
    hy = 2.f/(del*del) * (-rv*hh(rv) + 2.f * (rv - del/2.f)*hh(rv-del/2.f) -
             (rv-del)*hh(rv-del));
    hz = 2.f/(del*del) * (-rw*hh(rw) + 2.f * (rw - del/2.f)*hh(rw-del/2.f) -
             (rw-del)*hh(rw-del));

    /* now the time dependence (gaussian) */
    xpwidt = (PI * fpeak) * (PI * fpeak);
    gtp = exp( - (time*time) * xpwidt);
    gt = -2.f*xpwidt*(1.f-2.f*xpwidt*(time*time)) * gtp;

    /* finally, put it all together */
    dircsx = (ix*dx)/ru;
    dircsy = ((jy+0.5f)*dy)/rv;
    dircsz = ((kz+0.5f)*dz)/rw;

    *fsorsu = gt * hx * dircsx;
    *fsorsv = gt * hy * dircsy;
    *fsorsw = gt * hz * dircsz;
}
/* ============================================================== */
/*                                                                */
/*          Function hetiso.cs                                    */
/*                                                                */
/* ============================================================== */
void hetiso()
{
    mono int ltime;
    mono char lsors;

    void source(int ltime);
    void output_files(mono int ltime);

    lsors = TRUE;          /* input a source */

    /* boundary conditions can be set up outside time loop for parallel */
    /*       processing - set up for asorbing boundaries                */
    [domain model].{boundary();}

    /* time loop - start at time step nt1 = 3 */
    for (ltime = nt1; ltime <= nt; ltime++)
    {
        /* calculate the stress derivative terms */
        [domain model].{stress();}

        /* now, calculate displacements */
        [domain model].{calc_displ();}

         /* update the time series */
        [domain model].{update();}

        /* input the source */
        if(lsors)
            source(ltime);

        /* output options here */
        if (ltime % snap_file == 0)
            output_files(ltime);

         /* output to frame buffer */
         if (ltime % snap_fb == 0)
            [domain model].{output_fb();}

    }        /* end of time loop */
}

/* ================================================================ */
/*                                                                  */
/*           Function hh                                            */
/*                                                                  */
/* ================================================================ */
float hh(float r)
{
    if(r < 0.0)
        return(0.0f);
    else
    {
        if(r < 0.00001)
            return(0.5);
        else
            return(1.0);
    }
}
/* ================================================================ */
/*                                                                  */
/*          Function initialize - initialize the grid              */
/*                                                                 */
/* ============================================================== */
void model::initialize()
{
    u0 = u1 = u2 = 0.0;
    v0 = v1 = v2 = 0.0;
    w0 = w1 = w2 = 0.0;
}
/* ============================================================== */
/*                                                                 */
/*        Function input_defs                                      */
/*                                                                 */
/* ============================================================== */
void input_defs()
{
    FILE *fp;
    int nz,i,j,k,pr_indx;
    int nzones,nx1,ny1,nz1,nxn,nyn,nzn;
    float dt2,cclamb,ccmu,trho;

    /* open the file containing model parameters */
    if ((fp = fopen("xmodel.dat","r")) == NULL)
        printf("Error opening input file: xmodel.dat\n");

    /* read model parameters */
    fscanf(fp,"%d", &nt);
    printf("Number of time steps: %d\n\n", nt);
    fscanf(fp,"%f%f%f%f", &dx,&dy,&dz,&dt);
    fscanf(fp,"%d%d", &snap_file, &snap_fb);
    fscanf(fp,"%d%d%d%d", &out_fb,&out_xy,&out_yz,&out_xz);
    fscanf(fp,"%d%d%d%f", &isx,&isy,&isz,&fpeak);

    nt1 = 3;
    /* calculate lambda factors for use in ddispl */
    dt2 = dt*dt;
    xlamb = dt2/(dx*dx);
    ylamb = dt2/(dy*dy);
    zlamb = dt2/(dz*dz);
    xylamb = dt2/(dx*dy);
    xzlamb = dt2/(dx*dz);
    yzlamb = dt2/(dy*dz);

    /* read in number of zones of different elasticity, then read in
                constants for each zone */
    fscanf(fp,"%d", &nzones);

    for(nz=0; nz<nzones; nz++)
    {
        fscanf(fp,"%d%d%d%d%d%d", &nx1,&ny1,&nz1,&nxn,&nyn,&nzn);
        fscanf(fp,"%f%f%f", &cclamb,&ccmu,&trho);

        /* make the parallel variable assignments in parallel */
        [domain model].{
            int ix,iy,iz;

            ix = this_coordinate(0);
            iy = this_coordinate(1);
            iz = this_coordinate(2);

            /* select the active processors */
            if((ix >= nx1) && (ix <= nxn) &&
               (iy >= ny1) && (iy <= nyn) &&
               (iz >= nz1) && (iz <= nzn))
               {
                    clamb  = cclamb;
                    cmu    = ccmu;
                    clp2mu = cclamb + 2.f * ccmu;
                    rho    = trho;
               }    /* end if */
        }           /* end domain */
    }               /* end for */
}

/* ============================================================== */
/*                                                                */
/*       Function itoa - integer to ASCII conversion              */
/*                                                                */
/* ============================================================== */
void itoa(int n, char s[])
{
    int i, sign;
    void reverse(char s[]);

    if((sign = n) < 0)   /* record sign */
        n = -n;          /* make n positive */
    i = 0;
    do {
        s[i++] = n % 10 + '0';
    } while ((n /= 10) > 0);
    if (sign < 0)
        s[i++] = '-';
    s[i] = '\0';
    reverse(s);
}
/* ============================================================== */
/*                                                                  */
/*        Function output_fb - output to frame buffer               */
/*                                                                  */
/*        output options:                                           */
/*                out_fb = 0 for no frame buffer output             */
/*                       = 1 to display xy plane at ZDIM/2-1        */
/*                       = 2 to display yz plane at XDIM/2-1        */
/*                       = 3 to display xz plane at YDIM/2-1        */
/*                                                                  */
/* ================================================================ */
void model::output_fb(void)
{
    mono int nx2 = (XDIM/2)-1, ny2 = (YDIM/2)-1, nz2 = (ZDIM/2)-1;
    mono float slope, intercept;
    unsigned char color;
    int ix,iy,iz;

    /* calculate divergence and curl */
    divcurl();

    /* get minimum and maximum div and curl via reduction, use to */
    /*     scale for frame buffer                                  */
    slope = 23.0f / ((>?= div) - (<?= div));
    intercept = 1.0f - slope * (<?= div);

    /* calculate the color */
    color = slope * div + intercept;

    /* select processors to display */
    ix = this_coordinate(0);
    iy = this_coordinate(1);
    iz = this_coordinate(2);

    /* plot the xy plane */
    if(out_fb == 1)
    {
        if(iz == nz2)
            plot_x_y((unsigned short)ix, (unsigned short)iy, color);
    }

    /* plot the yz plane */
    if(out_fb == 2)
    {
        if(ix == nx2)
            plot_x_y((unsigned short)iy, (unsigned short)iz, color);
    }

    /* plot the xz plane */
    if(out_fb == 3)
    {
        if(iy == ny2)
            plot_x_y((unsigned short)ix, (unsigned short)iz, color);
    }
}
/* ============================================================ */
/*                                                              */
/*      Function output_files - write out ASCII disk files      */
/*                                                              */
/*      output options:                                         */
/*              xy_out = 1 to output div_xy and curlz           */
/*              yz_out = 1 to output div_yz and curlx           */
/*              xz_out = 1 to output div_xz and curly           */
/*                                                              */
/* ============================================================ */
void output_files(mono int ltime)
{
    mono int i,j,k,indx,indx2;
    mono int nx2 = (XDIM/2)-1, ny2 = (YDIM/2)-1, nz2 = (ZDIM/2)-1;
    char divxy[15], divyz[15], divxz[15];
    char curlx[15], curly[15], curlz[15];
    char tstep[4], extent[5];
    FILE *fp_divxz, *fp_divyz, *fp_divxy;
    FILE *fp_curlx, *fp_curly, *fp_curlz;
    char *strcat(char *file1, char *file2);
    char *strcpy(char *file1, char *file2);
    void itoa(int n, char *s);

    strcpy(extent,".dat");
    itoa(ltime, tstep);

    itoa(ltime, tstep);

    /* open ASCII output files for div and curl */
    if(out_xy)
    {
        strcpy(divxy,"divxy_");
        strcat(divxy, tstep);
        strcat(divxy, extent);
        printf("Opening file %s\n", divxy);
        if ((fp_divxy = fopen(divxy,"w")) == NULL)
             printf("Error opening output file: %s\n", divxy);
        strcpy(curlz,"curlz_");
        strcat(curlz, tstep);
        strcat(curlz, extent);
        printf("Opening file %s\n", curlz);
        if ((fp_curlz = fopen(curlz,"w")) == NULL)
    printf("Error opening output file: %s\n", curlz);
}
if(out_yz)
{
    strcpy(divyz,"divyz_");
    strcat(divyz, tstep);
    strcat(divyz, extent);
    printf("Opening file %s\n", divyz);
    if ((fp_divyz = fopen(divyz,"w")) == NULL)
         printf("Error opening output file: %s\n", divyz);
    strcpy(curlx,"curlx_");
    strcat(curlx, tstep);
    strcat(curlx, extent);
    printf("Opening file %s\n", curlx);
    if ((fp_curlx = fopen(curlx,"w")) == NULL)
         printf("Error opening output file: %s\n", curlx);
}
if(out_xz)
{
    strcat(divxz,"divxz_");
    strcat(divxz, tstep);
    strcat(divxz, extent);
    printf("Opening file %s\n", divxz);
    if ((fp_divxz = fopen(divxz,"w")) == NULL)
        printf("Error opening output file: %s\n", divxz);
    strcat(curly,"curly_");
    strcat(curly, tstep);
    strcat(curly, extent);
    printf("Opening file %s\n", curly);
    if ((fp_curly = fopen(curly,"w")) == NULL)
        printf("Error opening output file: %s\n", curly);
}

/* calculate divergence and curl */
[domain model].{divcurl();}

/* write out principal axis output plot files (snapshot output plots) */
if(out_xy)
{
    fprintf(fp_divxy,"divergence - xy plane at z = %d:\n", nz2);
    for(i = 0; i < XDIM; i++)
    for(j = 0; j < YDIM; j++)
    {
        indx = index_from_grid(i,nz2,j);
        fprintf(fp_divxy, "%9.2e ",point[indx].div);
        if(((j+1) % 8) == 0) fprintf(fp_divxy, "\n");
    }
    fprintf(fp_divxy, "\n");
    fprintf(fp_curlz, "curlz - xy plane at z = %d:\n", nz2);
    for(i = 0; i < XDIM; i++)
        for(j = 0; j < YDIM; j++)
        {
            indx = index_from_grid(i,nz2,j);
            fprintf(fp_curlz, "%9.2e ",point[indx].curlz);
            if(((j+1) % 8) == 0) fprintf(fp_curlz, "\n");
        }
    fprintf(fp_curlz, "\n");
}

if(out_yz)
{
    fprintf(fp_divyz, "divergence - yz plane at x = %d:\n", nx2);
    for(i = 0; i < YDIM; i++)
        for(j = 0; j < ZDIM; j++)
        {
            indx = index_from_grid(i,nx2,j);
            fprintf(fp_divyz, "%9.2e ",point[indx].div);
            if(((j+1) % 8) == 0) fprintf(fp_divyz, "\n");
        }
    fprintf(fp_divyz, "\n");

    fprintf(fp_curlx, "curlx - yz plane at x = %d:\n", nx2);
    for(i = 0; i < YDIM; i++)
        for(j = 0; j < ZDIM; j++)
        {
            indx = index_from_grid(i,nx2,j);
            fprintf(fp_curlx, "%9.2e ",point[indx].curlx);
            if(((j+1) % 8) == 0) fprintf(fp_curlx, "\n");
        }
    fprintf(fp_curlx, "\n");
}
if(out_xz)
{
    fprintf(fp_divxz, "divergence - xz plane at y = %d:\n", ny2);
    for(i = 0; i < XDIM; i++)
        for(j = 0; j < ZDIM; j++)
        {
            indx = index_from_grid(i,ny2,j);
            fprintf(fp_divxz, "%9.2e ",point[indx].div);
            if(((j+1) % 8) == 0) fprintf(fp_divxz, "\n");
        }
    fprintf(fp_divxz, "\n");

    fprintf(fp_curly, "curly - xz plane at y = %d:\n", ny2);
    for(i = 0; i < XDIM; i++)
```

```
        for(j = 0; j < ZDIM; j++)
        {
            indx = index_from_grid(i,ny2,j);
            fprintf(fp_curly, "%9.2e ",point[indx].curly);
            if(((j+1) % 8) == 0) fprintf(fp_curly, "\n");
        }
        fprintf(fp_curly, "\n");
    }
}

/* ============================================================ */
/*                                                              */
/*          Function reverse                                    */
/*                                                              */
/* ============================================================ */
void reverse(char s[])
{
    int c,i,j;
    int strlen(char *s);

    for (i = 0, j = strlen(s)-1; i<j; i++, j--)
    {
        c = s[i];
        s[i] = s[j];
        s[j] = c;
    }
}

/*============================================================= */
/*                                                              */
/*          Function setup_fb - initialize frame buffer         */
/*                 and define color table                       */
/*                                                              */
/*============================================================= */
void setup_fb(void)
{
    int i;
    /* initialize the frame buffer */
    init_frame_buffer(128,128);

    /* define the color table */
    for(i=0; i<25; i++)
        set_color(i+1, 240-i*10, i*10,i*10);
}
/* ========================================================= */
/*                                                           */
/*          Function source - updates the displacement values */
/*                  for the source by calling function force */
/*              which calculates the force representation of  */
/*              a point source.  The source location is given */
/*              by [isx][isy][isz].                           */
/*                                                            */
/* ========================================================== */
void source(int ltime)
{
  int i,j,k,ix,jy,kz,pt_indx;
  float xoff = 1.0e-10;
  float fsorsu,fsorsv,fsorsw;

  for (i = 0; i <= DEL; i++)
    {
    ix = 0 - i;
    for (j = 0; j <= DEL; j++)
    {
      jy = 0 - j;
      for (k = 0; k <= DEL; k++)
      {
        kz = 0 - k;
        if ((ix == 0) && (jy == 0) && (kz == 0))
        ;
        else
        {
          force(ix,jy,kz,ltime,&fsorsu,&fsorsv,&fsorsw);

          /* add the forcing term to displacements at corners of cubes */
          if ((ix != 0) && (jy != 0) && (kz != 0))
          {
             /* get the point index for the location to be calculated */
            pt_indx = index_from_grid(isx+ix,isy+jy,isz+kz);
            point[pt_indx].u1 = point[pt_indx].u1 + fsorsu;
            pt_indx = index_from_grid(isx+ix,isy-jy,isz+kz);
            point[pt_indx].u1 = point[pt_indx].u1 + fsorsu;
            pt_indx = index_from_grid(isx+ix,isy-jy,isz-kz);
            point[pt_indx].u1 = point[pt_indx].u1 + fsorsu;
            pt_indx = index_from_grid(isx+ix,isy+jy,isz-kz);
            point[pt_indx].u1 = point[pt_indx].u1 + fsorsu;
            pt_indx = index_from_grid(isx-ix,isy-jy,isz+kz);
            point[pt_indx].u1 = point[pt_indx].u1 - fsorsu;
            pt_indx = index_from_grid(isx-ix,isy-jy,isz-kz);
            point[pt_indx].u1 = point[pt_indx].u1 - fsorsu;
            pt_indx = index_from_grid(isx-ix,isy+jy,isz+kz);
            point[pt_indx].u1 = point[pt_indx].u1 - fsorsu;
            pt_indx = index_from_grid(isx-ix,isy+jy,isz-kz);
            point[pt_indx].u1 = point[pt_indx].u1 - fsorsu;

            pt_indx = index_from_grid(isx+ix,isy+jy,isz+kz);
    point[pt_indx].v1 = point[pt_indx].v1 + fsorsv;
    pt_indx = index_from_grid(isx+ix,isy-jy-1,isz+kz);
    point[pt_indx].v1 = point[pt_indx].v1 - fsorsv;
    pt_indx = index_from_grid(isx-ix-1,isy+jy,isz+kz);
    point[pt_indx].v1 = point[pt_indx].v1 + fsorsv;
    pt_indx = index_from_grid(isx-ix-1,isy-jy-1,isz+kz);
    point[pt_indx].v1 = point[pt_indx].v1 - fsorsv;
    pt_indx = index_from_grid(isx+ix,isy+jy,isz-kz);
    point[pt_indx].v1 = point[pt_indx].v1 + fsorsv;
    pt_indx = index_from_grid(isx+ix,isy-jy-1,isz-kz);
    point[pt_indx].v1 = point[pt_indx].v1 - fsorsv;
    pt_indx = index_from_grid(isx-ix-1,isy+jy,isz-kz);
    point[pt_indx].v1 = point[pt_indx].v1 + fsorsv;
    pt_indx = index_from_grid(isx-ix-1,isy-jy-1,isz-kz);
    point[pt_indx].v1 = point[pt_indx].v1 - fsorsv;

    pt_indx = index_from_grid(isx+ix,isy+jy,isz+kz);
    point[pt_indx].w1 = point[pt_indx].w1 + fsorsw;
    pt_indx = index_from_grid(isx+ix,isy+jy,isz-kz-1);
    point[pt_indx].w1 = point[pt_indx].w1 - fsorsw;
    pt_indx = index_from_grid(isx+ix,isy-jy,isz+kz);
    point[pt_indx].w1 = point[pt_indx].w1 + fsorsw;
    pt_indx = index_from_grid(isx+ix,isy-jy,isz-kz-1);
    point[pt_indx].w1 = point[pt_indx].w1 - fsorsw;
    pt_indx = index_from_grid(isx-ix-1,isy+jy,isz+kz);
    point[pt_indx].w1 = point[pt_indx].w1 + fsorsw;
    pt_indx = index_from_grid(isx-ix-1,isy+jy,isz-kz-1);
    point[pt_indx].w1 = point[pt_indx].w1 - fsorsw;
    pt_indx = index_from_grid(isx-ix-1,isy-jy,isz+kz);
    point[pt_indx].w1 = point[pt_indx].w1 + fsorsw;
    pt_indx = index_from_grid(isx-ix-1,isy-jy,isz-kz-1);
    point[pt_indx].w1 = point[pt_indx].w1 - fsorsw;

  }
  /* find values at the xy corners in the z-plane of the source */
  else
    if ((ix != 0) && (jy != 0))
    {
      pt_indx = index_from_grid(isx+ix,isy+jy,isz);
      point[pt_indx].u1 = point[pt_indx].u1 + fsorsu;
      pt_indx = index_from_grid(isx+ix,isy-jy,isz);
      point[pt_indx].u1 = point[pt_indx].u1 + fsorsu;
      pt_indx = index_from_grid(isx-ix,isy+jy,isz);
      point[pt_indx].u1 = point[pt_indx].u1 - fsorsu;
      pt_indx = index_from_grid(isx-ix,isy-jy,isz);
      point[pt_indx].u1 = point[pt_indx].u1 - fsorsu;

      pt_indx = index_from_grid(isx+ix,isy+jy,isz);
  point[pt_indx].v1 = point[pt_indx].v1 + fsorsv;
  pt_indx = index_from_grid(isx+ix,isy-jy-1,isz);
  point[pt_indx].v1 = point[pt_indx].v1 - fsorsv;
  pt_indx = index_from_grid(isx-ix-1,isy+jy,isz);
  point[pt_indx].v1 = point[pt_indx].v1 + fsorsv;
  pt_indx = index_from_grid(isx-ix-1,isy-jy-1,isz);
  point[pt_indx].v1 = point[pt_indx].v1 - fsorsv;
  /* no w-displacement updates needed here */
}

/* next the xz corners within the y-plane of the source */
else
  if ((ix != 0) && (kz != 0))
  {
    pt_indx = index_from_grid(isx+ix,isy,isz+kz);
    point[pt_indx].u1 = point[pt_indx].u1 + fsorsu;
    pt_indx = index_from_grid(isx+ix,isy,isz-kz);
    point[pt_indx].u1 = point[pt_indx].u1 + fsorsu;
    pt_indx = index_from_grid(isx-ix,isy,isz+kz);
    point[pt_indx].u1 = point[pt_indx].u1 - fsorsu;
    pt_indx = index_from_grid(isx-ix,isy,isz-kz);
    point[pt_indx].u1 = point[pt_indx].u1 - fsorsu;

    pt_indx = index_from_grid(isx+ix,isy,isz+kz);
    point[pt_indx].w1 = point[pt_indx].w1 + fsorsw;
    pt_indx = index_from_grid(isx+ix,isy,isz-kz-1);
    point[pt_indx].w1 = point[pt_indx].w1 - fsorsw;
    pt_indx = index_from_grid(isx-ix-1,isy,isz+kz);
    point[pt_indx].w1 = point[pt_indx].w1 + fsorsw;
    pt_indx = index_from_grid(isx-ix-1,isy,isz-kz-1);
    point[pt_indx].w1 = point[pt_indx].w1 - fsorsw;
    /* no v-displacement updates needed here */
  }

  /* note: if jy != 0 and kz != 0 (i.e. ix = 0) then no updates
           to displacement are needed since the v and w updates
           have already been handled in the previous sections
           (due to the staggered grid geometry and there is no
           u update because of symmetry. */
  else

  /* remaining updates are for the case of two zero values and one
           non-zero value for ix,jy,kz (but only ix != 0 is needed
           because of symmetry) */
    if ((ix != 0) && (jy == 0) && (kz == 0))
    {
      pt_indx = index_from_grid([illegible]+ix,isy,isz);
      point[pt_indx].u1 = point[[illegible]ndx].u1 + fsorsu;
```

```
                    pt_indx = index_from_grid(isx-ix,isy,isz);
                    point[pt_indx].u1 = point[pt_indx].u1 - fsorsu;

                    /* check on the magnitude of the 9-3,0,0) forsu term -
                       if abs(this term) < xoff then source is negligible
                       and is turned off.  Note: only check this after a full
                       cycle since don't want to shut off at zero crossing */
                    if(ltime > 25)
                      if(ix == -3)
                        if(fabs(fsorsu) < xoff)
                                lsors = FALSE;
                   }
              }
            }
          }
        }
}


/* ========================================================== */
/*                                                            */
/*          Function stress                                   */
/*                                                            */
/* ========================================================== */
void model::stress(void)
{
    int ix,iy,iz;

    ix = this_coordinate(0);
    iy = this_coordinate(1);
    iz = this_coordinate(2);

    if((ix < 1 || ix >= (XDIM-1)) || (iy < 1 || iy >= (YDIM-1)) ||
       (iz < 1 || iz >= (ZDIM-1)))
       ;
    else
    {
        calc_txxdx();
        calc_txydy();
        calc_txzdz();

        calc_txydx();
        calc_tyydy();
        calc_tyzdz();

        calc_txzdx();
        calc_tyzdy();
        calc_tzzdz();
    }
```

```
}
/* ========================================================== */
/*                                                            */
/*          Function update                                   */
/*                                                            */
/* ========================================================== */
void model::update(void)
{
    u2 = u1;
    v2 = v1;
    w2 = w1;

    u1 = u0;
    v1 = v0;
    w1 = w0;
}
/* ========================================================== */
/*                                                            */
/*      End of wave_3d functions                              */
/*                                                            */
/* ========================================================== */
/* ============================================================ */
/*      wave_3d.hs - include file for wave_3d.cs                */
/*                                                              */
/*      Programmer:  Julie Allen                                */
/*      Date:        28 February 1989                           */
/*                                                              */
/* ============================================================ */
/*                                                              */
/*        Copyright (c) 1989                                    */
/*        Information Processing and Communications Laboratory */
/*        Woods Hole Oceanographic Institution                  */
/*        All rights reserved.                                  */
/*                                                              */
/*        This material cannot be distributed or sold without   */
/*                prior permission of the author(s).            */
/*                                                              */
/* ============================================================ */

/* define macros for timing */
#define TIMER_ON        CM_start_timer(1)
#define TIMER_RESET     CM_reset_timer()
#define TIMER_OFF       (void) CM_stop_timer(1)

/* common definitions for 3D finite difference modeling          */
#define         XDIM            64              /* model dimensions */
#define         YDIM            64
#define         ZDIM            64
#define         TOTAL           XDIM*YDIM*ZDIM
#define         NTEL            4
#define         DEL             6
#define         PI              3.14159265f
#define         TRUE            1
#define         FALSE           0
#define         SET_DEBUG       FALSE

/* define common variables */
mono char out_fb = FALSE, out_xy = FALSE, out_yz = FALSE, out_xz = FALSE;
mono int snap_file,snap_fb,nt,nt1;
mono float xlamb,ylamb,zlamb,xylamb,xzlamb,yzlamb;
mono float dx,dy,dz,dt,fpeak;
mono int isx,isy,isz;
mono char lsors, DEBUG = FALSE;

/* define the domain for the parallel variables - 27 variables = 108 bytes */
domain model {
    float u0,u1,u2;
    float v0,v1,v2;
    float w0,w1,w2;
    float clamb;
    float cmu;
    float clp2mu;
    float rho;
    float txxdx,txydy,txzdz;
    float txydx,tyydy,tyzdz;
    float txzdx,tyzdy,tzzdz;
    float alpha;
    float div,curlx,curly,curlz;
    float alpha_fct(int indx);
    void boundary(void);
    void calc_adisp(float *adisp);
    void calc_bdisp(float *bdisp);
    void calc_cdisp(float *cdisp);
    void calc_displ(void);
    void calc_txxdx(void);
    void calc_txydy(void);
    void calc_txzdz(void);
    void calc_txydx(void);
    void calc_tyydy(void);
    void calc_tyzdz(void);
    void calc_txzdx(void);
    void calc_tyzdy(void);
    void calc_tzzdz(void);
    void divcurl(void);
    void initialize(void);
    void output_fb(void);
    void stress(void);
    void update(void);
} point[TOTAL];
/* ============================================================ */
```

D.2 convert program

```
c===============================================================
        program convert
c
c       Programmer: Julie Allen
c       Date:       16 June 1989
c
c       Copyright (c) 1989
c       Information Processing and Communications Laboratory
c       Woods Hole Oceanographic Institution
c       All rights reserved.
c
c       This material cannot be distributed or sold without
c           prior permission of the author(s).
c
c===============================================================
c       Program is currently hardwired for models with dimensions
c                64x64x64
c
c       Convert formatted files output on CM and transferred via ftp to
c                the VAX to binary files ready for input to program SWAP
c
c===============================================================
        real*4 value(64,64)
        integer*4 infile,ioutfile
c
        data infile/11/,ioutfile/12/
c
 1001   format(a)
c
        print '(1x,a,$)','Enter scale factor (1 for no scaling): '
        read *,scale
c
c       Open input file
c
        open(infile,status='old',form='formatted')
c
c       Open output file
c
        open(ioutfile,status='new',form='unformatted')
c
        read(infile,1001,err=800,end=900) dummy
        do i=1,64
            do j = 1,64,8
                read(infile,*,err=800,end=900) (value(i,k),k=j,j+7)
            enddo
        enddo
```

```
c
c       Scale the data
c
        if(scale.ne.1.0) then
            valmin = 1000.0
            valmax = -1000.0
            do i=1,64
                do j=1,64
                    value(i,j) = value(i,j) * scale
                    if(value(i,j).lt.valmin) valmin = value(i,j)
                    if(value(i,j).gt.valmax) valmax = value(i,j)
                enddo
            enddo
            print *,'minimum: ',valmin,'  maximum: ',valmax
        endif
c
        write(ioutfile)
        write(ioutfile)
        write(ioutfile) ((value(i,j),i=1,64),j=1,64)
c
        go to 900
c
  800   call errsns(ierr,,,iunit)
        print *,'errsns # ',ierr,' on unit ',iunit
c
  900   close(infile)
        close(ioutfile)
        stop
        end
c================================================================
```

D.3 n-dimensional grid library

The following source code, used in the **wave_3d** program, was written by Robert Whaley, Site Representative from TMC at the NRL Connection Machine Facility.

```
/* nd_grid - n-dimensional grid functions     */
/*                                            */
/* Author:      Robert Whaley                 */
/* E-mail:      whaley@nrl-cmf.arpa           */
/* US mail:     Robert Whaley                 */
/*              Naval Research Lab            */
/*              Code 5150                     */
/*              Washington, D.C.  20375-5000 */
/* Phone:       (202) 404-7019                */
/* Affiliate:   TMC  (Site Rep at NRL)        */

#include <cm/cm.hs>
#include <stdio.hs>
#include "nd_grid.hs"

#define bit_sizeof(x) (8 * sizeof(x))

void make_grid(int dummy) {
  printf("Error: make_grid called from mono context.")
  printf("Call make_grid from domain context\n");
}

static void make_grid_internal(int dim_count,
                       unsigned dimension_array[]) {
  int dims, total_procs;
  CM_geometry_id_t a_geometry;
  a_geometry = CM_create_geometry(dimension_array,dim_count);
  CM_set_vp_set_geometry(CM_current_vp_set, a_geometry);
  total_procs = 1;
  for (dims=0; dims<dim_count; dims++)
    total_procs *= dimension_array[dims];
  if (total_procs != CM_global_count_context()){
    printf("The number of active processors: %d ",
CM_global_count_context());
    printf("is not equal to the total grid size: %d\n",
total_procs);
  }
}

void void::make_grid(mono int x) {
  mono unsigned dimension_array[1];
  dimension_array[0] = x;
  make_grid_internal(1,dimension_array);
}

void void::make_grid(mono int x, mono int y) {
  mono unsigned dimension_array[2];
  dimension_array[0] = x;
  dimension_array[1] = y;
  make_grid_internal(2,dimension_array);
}

void void::make_grid(mono int x, mono int y, mono int z) {
  mono unsigned dimension_array[3];
  dimension_array[0] = x;
  dimension_array[1] = y;
  dimension_array[2] = z;
  make_grid_internal(3,dimension_array);
}

void void::make_grid(mono int x, mono int y, mono int z, mono int a)
{
  mono unsigned dimension_array[4];
  dimension_array[0] = x;
  dimension_array[1] = y;
  dimension_array[2] = z;
  dimension_array[3] = a;
  make_grid_internal(4,dimension_array);
}

void void::make_grid(mono int x, mono int y, mono int z, mono int a,
                     mono int b) {
  mono unsigned dimension_array[5];
  dimension_array[0] = x;
  dimension_array[1] = y;
  dimension_array[2] = z;
  dimension_array[3] = a;
  dimension_array[4] = b;
  make_grid_internal(5,dimension_array);
}

void void::make_grid(mono int x, mono int y, mono int z, mono int a,
                     mono int b, mono int c) {
  mono unsigned dimension_array[6];
  dimension_array[0] = x;
  dimension_array[1] = y;
  dimension_array[2] = z;
  dimension_array[3] = a;
  dimension_array[4] = b;
  dimension_array[5] = c;
  make_grid_internal(6,dimension_array);
}
```

```
void void::make_grid(mono int x, mono int y, mono int z, mono int a,
                     mono int b, mono int c, mono int d) {
  mono unsigned dimension_array[7];
  dimension_array[0] = x;
  dimension_array[1] = y;
  dimension_array[2] = z;
  dimension_array[3] = a;
  dimension_array[4] = b;
  dimension_array[5] = c;
  dimension_array[6] = d;
  make_grid_internal(7,dimension_array);
}

void void::make_grid(mono int x, mono int y, mono int z, mono int a,
                     mono int b, mono int c, mono int d, mono int e)
{
  mono unsigned dimension_array[8];
  dimension_array[0] = x;
  dimension_array[1] = y;
  dimension_array[2] = z;
  dimension_array[3] = a;
  dimension_array[4] = b;
  dimension_array[5] = c;
  dimension_array[6] = d;
  dimension_array[7] = e;
  make_grid_internal(8,dimension_array);
}

void void::make_hypercube_grid() {
  mono unsigned dimension_array[32];
  mono int dim_count, i;
  mono int proc_count = CM_geometry_total_processors(
                     CM_vp_set_geometry(CM_current_vp_set));
  for (dim_count = 0; proc_count >>= 1; dim_count++);
  for (i=0;i<dim_count;i++) dimension_array[i] = 2;
  make_grid_internal(dim_count,dimension_array);
}

int void::this_coordinate(mono int dim) {
  int my_grid_address;
  CM_my_news_coordinate_1L(&my_grid_address, dim,
      bit_sizeof(my_grid_address));
  return(my_grid_address);}
```

```
int void::next(mono int dim, int void:: * mono value_to_get) {
  int return_value;
  mono CM_memaddr_t kludge = value_to_get;
  CM_get_from_news_1L(&return_value, kludge, dim, CM_upward,
                      bit_sizeof(return_value));
  return(return_value);}

unsigned void::next(mono int dim, unsigned void:: * mono value_to_get)
{
  unsigned return_value;
  mono CM_memaddr_t kludge = value_to_get;
  CM_get_from_news_1L(&return_value, kludge, dim, CM_upward,
                      bit_sizeof(return_value));
  return(return_value);}

float void::next(mono int dim, float void:: * mono value_to_get) {
  float return_value;
  mono CM_memaddr_t kludge = value_to_get;
  CM_get_from_news_1L(&return_value, kludge, dim, CM_upward,
                      bit_sizeof(return_value));
  return(return_value);}

int void::prev(mono int dim, int void:: * mono value_to_get) {
  int return_value;
  mono CM_memaddr_t kludge = value_to_get;
  CM_get_from_news_1L(&return_value, kludge, dim, CM_downward,
                      bit_sizeof(return_value));
  return(return_value);}

unsigned void::prev(mono int dim, unsigned void:: * mono value_to_get)
{
  unsigned return_value;
  mono CM_memaddr_t kludge = value_to_get;
  CM_get_from_news_1L(&return_value, kludge, dim, CM_downward,
                      bit_sizeof(return_value));
  return(return_value);}

float void::prev(mono int dim, float void:: * mono value_to_get) {
  float return_value;
  mono CM_memaddr_t kludge = value_to_get;
  CM_get_from_news_1L(&return_value, kludge, dim, CM_downward,
                      bit_sizeof(return_value));
  return(return_value);}
```

```
int mono_index_from_grid_internal(int dim_count, int *dims) {
  int i;
  int processor_address;
  CM_geometry_id_t a_geometry;
  a_geometry = CM_vp_set_geometry(CM_current_vp_set);
  processor_address = CM_fe_make_news_coordinate(a_geometry, 0, dims[0]);
  for (i = 1; i < dim_count; i++)
    processor_address = CM_fe_deposit_news_coordinate(
                          a_geometry, processor_address, i, dims[i]);
  return(processor_address);
}

int index_from_grid(int x) {
  return(mono_index_from_grid_internal(1,&x));
}

int index_from_grid(int x, int y) {
  int dim_array[2];
  dim_array[0] = x;
  dim_array[1] = y;
  return(mono_index_from_grid_internal(2,dim_array));
}

int index_from_grid(int x, int y, int z) {
  int dim_array[3];
  dim_array[0] = x;
  dim_array[1] = y;
  dim_array[2] = z;
  return(mono_index_from_grid_internal(3,dim_array));
}

int index_from_grid(int x, int y, int z, int a) {
  int dim_array[4];
  dim_array[0] = x;
  dim_array[1] = y;
  dim_array[2] = z;
  dim_array[3] = a;
  return(mono_index_from_grid_internal(4,dim_array));
}

int index_from_grid(int x, int y, int z, int a,
                                      int b) {
  int dim_array[5];
  dim_array[0] = x;
  dim_array[1] = y;
  dim_array[2] = z;
  dim_array[3] = a;
  dim_array[4] = b;
  return(mono_index_from_grid_internal(5,dim_array));
}

int index_from_grid(int x, int y, int z, int a,
                                      int b, int c) {
  int dim_array[6];
  dim_array[0] = x;
  dim_array[1] = y;
  dim_array[2] = z;
  dim_array[3] = a;
  dim_array[4] = b;
  dim_array[5] = c;
  return(mono_index_from_grid_internal(6,dim_array));
}

int index_from_grid(int x, int y, int z, int a,
                                      int b, int c, int d) {
  int dim_array[7];
  dim_array[0] = x;
  dim_array[1] = y;
  dim_array[2] = z;
  dim_array[3] = a;
  dim_array[4] = b;
  dim_array[5] = c;
  dim_array[6] = d;
  return(mono_index_from_grid_internal(7,dim_array));
}

int index_from_grid(int x, int y, int z, int a,
                                      int b, int c, int d, int e) {
  int dim_array[8];
  dim_array[0] = x;
  dim_array[1] = y;
  dim_array[2] = z;
  dim_array[3] = a;
  dim_array[4] = b;
  dim_array[5] = c;
  dim_array[6] = d;
  dim_array[7] = e;
  return(mono_index_from_grid_internal(8,dim_array));
}
```

```
static CM_geometry_id_t a_geometry_for_m_p_f_g_i;

int void::index_from_grid_internal(mono int dim_count, int * mono dims)
{
  mono int i;
  int processor_address;
  a_geometry_for_m_p_f_g_i = CM_vp_set_geometry(CM_current_vp_set);
  CM_make_news_coordinate_1L(a_geometry_for_m_p_f_g_i, &processor_address,
                          0, dims, bit_sizeof(processor_address));
  for (i = 1; i < dim_count; i++)
    CM_deposit_news_coordinate_1L(a_geometry_for_m_p_f_g_i, &processor_address,
                                  i, dims+i, bit_sizeof(processor_address));
  return(processor_address);
}

int void::index_from_grid(int x) {
  return(index_from_grid_internal(1,&x));
}

int void::index_from_grid(int x, int y) {
  int dim_array[2];
  dim_array[0] = x;
  dim_array[1] = y;
  return(index_from_grid_internal(2,dim_array));
}

int void::index_from_grid(int x, int y, int z) {
  int dim_array[3];
  dim_array[0] = x;
  dim_array[1] = y;
  dim_array[2] = z;
  return(index_from_grid_internal(3,dim_array));
}

int void::index_from_grid(int x, int y, int z, int a) {
  int dim_array[4];
  dim_array[0] = x;
  dim_array[1] = y;
  dim_array[2] = z;
  dim_array[3] = a;
  return(index_from_grid_internal(4,dim_array));
}

int void::index_from_grid(int x, int y, int z, int a,
                                        int b) {
  int dim_array[5];
  dim_array[0] = x;
  dim_array[1] = y;
  dim_array[2] = z;
  dim_array[3] = a;
  dim_array[4] = b;
  return(index_from_grid_internal(5,dim_array));
}

int void::index_from_grid(int x, int y, int z, int a,
                                        int b, int c) {
  int dim_array[6];
  dim_array[0] = x;
  dim_array[1] = y;
  dim_array[2] = z;
  dim_array[3] = a;
  dim_array[4] = b;
  dim_array[5] = c;
  return(index_from_grid_internal(6,dim_array));
}

int void::index_from_grid(int x, int y, int z, int a,
                                        int b, int c, int d) {
  int dim_array[7];
  dim_array[0] = x;
  dim_array[1] = y;
  dim_array[2] = z;
  dim_array[3] = a;
  dim_array[4] = b;
  dim_array[5] = c;
  dim_array[6] = d;
  return(index_from_grid_internal(7,dim_array));
}

int void::index_from_grid(int x, int y, int z, int a,
                                        int b, int c, int d, int e) {
  int dim_array[8];
  dim_array[0] = x;
  dim_array[1] = y;
  dim_array[2] = z;
  dim_array[3] = a;
  dim_array[4] = b;
  dim_array[5] = c;
  dim_array[6] = d;
  dim_array[7] = e;
  return(index_from_grid_internal(8,dim_array));
}
```

```
\* nd_grid.hs - include file for n-dimensional grid library */

domain void * pointer_from_grid(int x) {
  return(CUBEADDR_TO_POINTER(domain void, index_from_grid(x)));
}

domain void * pointer_from_grid(int x, int y) {
  return(CUBEADDR_TO_POINTER(domain void, index_from_grid(x,y)));
}

domain void * pointer_from_grid(int x, int y, int z) {
  return(CUBEADDR_TO_POINTER(domain void, index_from_grid(x,y,z)));
}

domain void * pointer_from_grid(int x, int y, int z, int a) {
  return(CUBEADDR_TO_POINTER(domain void,
         index_from_grid(x,y,z,a)));
}

domain void * pointer_from_grid(int x, int y, int z, int a,
                                int b) {
  return(CUBEADDR_TO_POINTER(domain void,
         index_from_grid(x,y,z,a,b)));
}

domain void * pointer_from_grid(int x, int y, int z, int a,
                                int b, int c) {
  return(CUBEADDR_TO_POINTER(domain void,
         index_from_grid(x,y,z,a,b,c)));
}

domain void * pointer_from_grid(int x, int y, int z, int a,
                                int b, int c, int d) {
  return(CUBEADDR_TO_POINTER(domain void,
         index_from_grid(x,y,z,a,b,c,d)));
}

domain void * pointer_from_grid(int x, int y, int z, int a,
                                int b, int c, int d, int e) {
  return(CUBEADDR_TO_POINTER(domain void,
         index_from_grid(x,y,z,a,b,c,d,e)));
}
```

```
domain void * void::pointer_from_grid(int x) {
  return(CUBEADDR_TO_POINTER(domain void, index_from_grid(x)));
}

domain void * void::pointer_from_grid(int x, int y) {
  return(CUBEADDR_TO_POINTER(domain void, index_from_grid(x,y)));
}

domain void * void::pointer_from_grid(int x, int y, int z) {
  return(CUBEADDR_TO_POINTER(domain void, index_from_grid(x,y,z)));
}

domain void * void::pointer_from_grid(int x, int y, int z, int a) {
  return(CUBEADDR_TO_POINTER(domain void,
         index_from_grid(x,y,z,a)));
}

domain void * void::pointer_from_grid(int x, int y, int z, int a,
                                      int b) {
  return(CUBEADDR_TO_POINTER(domain void,
         index_from_grid(x,y,z,a,b)));
}

domain void * void::pointer_from_grid(int x, int y, int z, int a,
                                      int b, int c) {
  return(CUBEADDR_TO_POINTER(domain void,
         index_from_grid(x,y,z,a,b,c)));
}

domain void * void::pointer_from_grid(int x, int y, int z, int a,
                                      int b, int c, int d) {
  return(CUBEADDR_TO_POINTER(domain void,
         index_from_grid(x,y,z,a,b,c,d)));
}

domain void * void::pointer_from_grid(int x, int y, int z, int a,
                                      int b, int c, int d, int e) {
  return(CUBEADDR_TO_POINTER(domain void,
         index_from_grid(x,y,z,a,b,c,d,e)));
}
```

```
overload make_grid, void::make_grid;
overload void::next, void::prev;
overload pointer_from_grid, void::pointer_from_grid;
overload index_from_grid, void::index_from_grid;
void make_grid(int, ...);    /* don't call this one, please */

void void::make_grid(mono int);
void void::make_grid(mono int, mono int);
void void::make_grid(mono int, mono int, mono int);
void void::make_grid(mono int, mono int, mono int, mono int);
void void::make_grid(mono int, mono int, mono int, mono int, mono int);
void void::make_grid(mono int, mono int, mono int, mono int, mono int,
                     mono int);
void void::make_grid(mono int, mono int, mono int, mono int, mono int,
                     mono int, mono int);
void void::make_grid(mono int, mono int, mono int, mono int, mono int,
                     mono int, mono int, mono int);

void void::make_hypercube_grid(void);

int void::this_coordinate(mono int);

int void::next(mono int dim, int * mono value_ptr);
int void::prev(mono int dim, int * mono value_ptr);
unsigned void::next(mono int dim, unsigned * mono value_ptr);
unsigned void::prev(mono int dim, unsigned * mono value_ptr);
float void::next(mono int dim, float * mono value_ptr);
float void::prev(mono int dim, float * mono value_ptr);

domain void * pointer_from_grid(int);
domain void * pointer_from_grid(int, int);
domain void * pointer_from_grid(int, int, int);
domain void * pointer_from_grid(int, int, int, int);
domain void * pointer_from_grid(int, int, int, int, int);
domain void * pointer_from_grid(int, int, int, int, int, int);
domain void * pointer_from_grid(int, int, int, int, int, int,
                                int);
domain void * pointer_from_grid(int, int, int, int, int, int,
                                int, int);
```

```
domain void * void::pointer_from_grid(int);
domain void * void::pointer_from_grid(int, int);
domain void * void::pointer_from_grid(int, int, int);
domain void * void::pointer_from_grid(int, int, int, int);
domain void * void::pointer_from_grid(int, int, int, int, int);
domain void * void::pointer_from_grid(int, int, int, int, int, int);
domain void * void::pointer_from_grid(int, int, int, int, int, int,
                                      int);
domain void * void::pointer_from_grid(int, int, int, int, int, int,
                                      int, int);

int index_from_grid(int);
int index_from_grid(int, int);
int index_from_grid(int, int, int);
int index_from_grid(int, int, int, int);
int index_from_grid(int, int, int, int, int);
int index_from_grid(int, int, int, int, int, int);
int index_from_grid(int, int, int, int, int, int, int);
int index_from_grid(int, int, int, int, int, int, int, int);

int void::index_from_grid(int);
int void::index_from_grid(int, int);
int void::index_from_grid(int, int, int);
int void::index_from_grid(int, int, int, int);
int void::index_from_grid(int, int, int, int, int);
int void::index_from_grid(int, int, int, int, int, int);
int void::index_from_grid(int, int, int, int, int, int, int);
int void::index_from_grid(int, int, int, int, int, int, int, int);
```

```
/* fb - C* frame buffer functions                  */
/*                                                 */
/* Author:       Robert Whaley                     */
/* E-mail:       whaley@nrl-cmf.arpa               */
/* US mail:      Robert Whaley                     */
/*               Naval Research Lab                */
/*               Code 5150                         */
/*               Washington, D.C.  20375-5000 */
/* Phone:        (202) 404-7019                    */
/* Affiliate:    TMC  (Site Rep at NRL)            */

#include <stdio.hs>
#include <cm/paris.hs>
#include <cm/cmfb.hs>
#include "nd_grid.hs"
#include "fb.hs"

#define bit_sizeof(x) (8 * sizeof(x))

static struct CMFB_display_id my_display;
static CM_vp_set_id_t my_vp_set;
static CM_vp_set_id_t old_vp_set;
static CM_vp_set_id_t same_geo_vp_set;
static CM_geometry_id_t my_geometry;
static CM_memaddr_t my_color;

int CMFB_width(struct CMFB_display_id *);
int CMFB_height(struct CMFB_display_id *);
CMFB_buffer_id_t CMFB_spare_buffer(struct CMFB_display_id *);
void CMFB_switch_buffer(struct CMFB_display_id *, CM_buffer_id_t);
unsigned _CMI_get_field_address_from_field_id_safely(CM_field_id_t);
unsigned _CMI_field_location(unsigned);
void _CMI_physical_move_always(unsigned, unsigned, unsigned);

char *getenv(char * name);

void init_frame_buffer(int x_dim, int y_dim) {
  unsigned dims[2];
  int physical_x, physical_y;
  int zoom;

  same_geo_vp_set = CM_allocate_vp_set(CM_vp_set_geometry(CM_current_vp_set));
  dims[0] = x_dim;
  dims[1] = y_dim;
  my_geometry = CM_create_geometry(dims,2);
  my_vp_set = CM_allocate_vp_set(my_geometry);

  if (CMFB_attach_display(getenv("CM_FRAMEBUFFER"), &my_display)) {
    if (getenv("CM_FRAMEBUFFER")) {
      printf("init_frame_buffer: Could not attach to frame buffer:
%s.\n",
             getenv("CM_FRAMEBUFFER"));
      printf("Framebuffer is probably not connected to your sequencers.\n");
    }
    else {
      printf("init_frame_buffer: Could not attach frame buffer.\n");
      printf("Probably using sequencers that have no frame buffer.\n");
    }
  }
  CMFB_initialize_display(&my_display, 8, 1);
  CMFB_initialize_color_table(&my_display);
  physical_x = CMFB_width(&my_display);
  physical_y = CMFB_height(&my_display);
  zoom = ((physical_x / x_dim) <? (physical_y / y_dim)) - 1;
  CMFB_set_zoom(&my_display, zoom, zoom, 0);
  old_vp_set = CM_current_vp_set;
  CM_set_vp_set(my_vp_set);
  my_color = (CM_memaddr_t) CM_allocate_heap_field(8);
  CM_u_move_zero_always_1L(my_color, 8);
  CM_set_vp_set(old_vp_set);
}

void release_frame_buffer() {
  CMFB_detach_display(&my_display);
  CM_deallocate_heap_field(my_color);
  CM_deallocate_vp_set(my_vp_set);
  CM_deallocate_geometry(my_geometry);
}

void void::plot_x_y(unsigned short x, unsigned short y,
                              unsigned char color) {
  mono CM_vp_set_id_t old_vp_set;
  mono CMFB_buffer_id_t the_buffer;
  unsigned int a_send_address;
  old_vp_set = CM_current_vp_set;
  CM_set_vp_set(my_vp_set);
  CM_u_move_zero_always_1L(my_color, 8);
  CM_set_vp_set(old_vp_set);
  CMFB_shuffle_from_x_y(&a_send_address, &x, &y, my_geometry);
  CM_send_1L(my_color, &a_send_address, &color, bit_sizeof(color),
                                (CM_field_id_t)CM_do_not_notify_token());
  CM_set_vp_set(my_vp_set);
  the_buffer = CMFB_spare_buffer(&my_display);
  CMFB_write_preshuffled_always(&my_display, the_buffer, my_color,
0, 0);
  CMFB_switch_buffer(&my_display, the_buffer);
  CM_set_vp_set(old_vp_set);
}

void void::plot_x_y_over(unsigned short x, unsigned short y,
                         unsigned char color) {
  mono CM_vp_set_id_t old_vp_set;
  mono CMFB_buffer_id_t the_buffer;
  unsigned int a_send_address;
  old_vp_set = CM_current_vp_set;
  CM_set_vp_set(my_vp_set);
  CM_set_vp_set(old_vp_set);
  CMFB_shuffle_from_x_y(&a_send_address, &x, &y, my_geometry);
  CM_send_1L(my_color, &a_send_address, &color, bit_sizeof(color),
                                (CM_field_id_t)CM_do_not_notify_token());
  CM_set_vp_set(my_vp_set);
  the_buffer = CMFB_spare_buffer(&my_display);
  CMFB_write_preshuffled_always(&my_display, the_buffer, my_color,
0, 0);
  CMFB_switch_buffer(&my_display, the_buffer);
  CM_set_vp_set(old_vp_set);
}

void void::plot_from_grid(unsigned char color) {
  mono CMFB_buffer_id_t the_buffer;
  mono int vp_index;
  void void:: * mono color_prime;
  mono unsigned color_phys_loc, color_prime_phys_loc;
  color_phys_loc = (unsigned) &color;
  old_vp_set = CM_current_vp_set;
  CM_set_vp_set(same_geo_vp_set);
  vp_index = CM_geometry_total_vp_ratio(CM_vp_set_geometry(old_vp_set));
  color_prime = CM_allocate_stack_field(8);
  color_prime_phys_loc =
    _CMI_field_location(
        _CMI_get_field_address_from_field_id_safely(color_prime));
  while (vp_index-- > 0) {
    _CMI_physical_move_always(color_prime_phys_loc, color_phys_loc,
8);
    color_prime_phys_loc += 8;
    color_phys_loc += (unsigned) CM_user_memory_address_limit + 4;
  }
  the_buffer = CMFB_spare_buffer(&my_display);
  CMFB_write_always(&my_display, the_buffer, color_prime, 0, 0);
  CMFB_switch_buffer(&my_display, the_buffer);
  CM_deallocate_stack_through(color_prime);
  CM_set_vp_set(old_vp_set);
}
```

```
/* fb.hs - include file for fb library */
void set_color(int color_id, int red, int green, int blue) {
  CMFB_write_color(&my_display, CMFB_red, color_id, red);
  CMFB_write_color(&my_display, CMFB_green, color_id, green);
  CMFB_write_color(&my_display, CMFB_blue, color_id, blue);
}
void init_frame_buffer(int x_dim, int y_dim);

void release_frame_buffer(void);

void void::plot_x_y(unsigned short x, unsigned short y,
                             unsigned char color);

void void::plot_x_y_over(unsigned short x, unsigned short y,
                             unsigned char color);

void void::plot_from_grid(unsigned char color);

void set_color(int color_id, int red, int green, int blue);
```

References

Alterman, Z. and F. C. Karal, 1968, Propagation of elastic waves in layered media by finite difference methods, BSSA, v. 58, 367-398.

Charrette, E. E., 1987, Three dimensional finite difference modeling on a very fine-grian parallel computer, in Annual Report of MIT Earth Resources Laboratore Reservoir Delineation Consortium.

Dougherty, Martin E. and Ralph A. Stephen, 1988, Seismic Energy Partitioning and Scattering in Laterally Heterogeneous Ocean Crust, PAGEOPH, Vol 128, 195-229.

Etgen, J. and K. Yomagida, 1988, Three dimensional wave propagation in the Los Angeles Basin, abstract, EOS, v. 69, no. 44, p 1325.

Fornberg, B., 1987, The pseudospectral method: comparisons with finite differences for the elastic wave equation, Geophysics, v. 41, p. 2-27.

Frankel, A. and R. W. Clayton, 1986, Finite difference simulations of seismic scattering: Implications for the propagation of short-period seismic waves in the crust and models of crustal heterogeneity, J. Geophys. Res. 91, 6465-6489.

Hillis, W. Daniel, 1987, The Connection Machine, Scientific American (256)6, 108-115.

Hunt, Mary M. and Ralph H. Stephen, 1986, A user's manual for finite difference synthetic seismogram codes on the CYBER 205 and CRAY XMP-12, W.H.O.I. Technical Memorandum No. 4-86.

Kelly, K. R., R. W. Ward, S. Treitel, and R. M. Alford, 1976, Synthetic seismograms: A finite difference approach, Geophysics 41, 2-27.

Levander, A. R., 1985, Use of the telegraphy equation to improve absorbing boundary efficiency for fourth-order acoustic wave finite difference schemes, Bull. Seism. Soc. Am. 75, 1847-1852.

Nicoletis, L., 1981, Simulation numerique del la propagation d'ondes sismiques dans les milieux stratifies a deux et trois dimensions: contributions a la construction et a l'interpretation des sismogrammes synthetiques, Ph.D. thesis, Universite Pierre et Marie Curie, Paris, France.

Stephen, R. A., 1983, A comparison of finite difference and reflectivity seismograms for marine models, Geophys. J. R. astr. Soc. 72, 39-58.

Stephen, R.A., 1984, Finite difference seismograms for laterally varying marine models, Geophys. J. R. astr. Soc., 79, 185-198.

Stephen, R. A., F. Pardo-Casas, and C. H. Cheng, 1985, Finite difference synthetic acoustic logs, Geophysics 50, 1588-1609.

Toksoz, M. N., A. M. Dainty and E. E. Charrette, 1988, Spatial variation of ground motion due to lateral heterogeneity, Proc. Internat. Workshop on Spatial variation of earthquake ground motion, in press.

Virieux, J., 1986, P-SV wave propagation in heterogeneous media: Velocity-stress finite difference method, Geophysics 51, 889-901.

DOCUMENT LIBRARY

July 5, 1989

Distribution List for Technical Report Exchange

Attn: Stella Sanchez-Wade
Documents Section
Scripps Institution of Oceanography
Library, Mail Code C-075C
La Jolla, CA 92093

Hancock Library of Biology & Oceanography
Alan Hancock Laboratory
University of Southern California
University Park
Los Angeles, CA 90089-0371

Gifts & Exchanges
Library
Bedford Institute of Oceanography
P.O. Box 1006
Dartmouth, NS, B2Y 4A2, CANADA

Office of the International Ice Patrol
c/o Coast Guard R & D Center
Avery Point
Groton, CT 06340

Library
Physical Oceanographic Laboratory
Nova University
8000 N. Ocean Drive
Dania, FL 33304

NOAA/EDIS Miami Library Center
4301 Rickenbacker Causeway
Miami, FL 33149

Library
Skidaway Institute of Oceanography
P.O. Box 13687
Savannah, GA 31416

Institute of Geophysics
University of Hawaii
Library Room 252
2525 Correa Road
Honolulu, HI 96822

Library
Chesapeake Bay Institute
4800 Atwell Road
Shady Side, MD 20876

MIT Libraries
Serial Journal Room 14E-210
Cambridge, MA 02139

Director, Ralph M. Parsons Laboratory
Room 48-311
MIT
Cambridge, MA 02139

Marine Resources Information Center
Building E38-320
MIT
Cambridge, MA 02139

Library
Lamont-Doherty Geological Observatory
Colombia University
Palisades, NY 10964

Library
Serials Department
Oregon State University
Corvallis, OR 97331

Pell Marine Science Library
University of Rhode Island
Narragansett Bay Campus
Narragansett, RI 02882

Working Collection
Texas A&M University
Dept. of Oceanography
College Station, TX 77843

Library
Virginia Institute of Marine Science
Gloucester Point, VA 23062

Fisheries-Oceanography Library
151 Oceanography Teaching Bldg.
University of Washington
Seattle, WA 98195

Library
R.S.M.A.S.
University of Miami
4600 Rickenbacker Causeway
Miami, FL 33149

Maury Oceanographic Library
Naval Oceanographic Office
Bay St. Louis
NSTL, MS 39522-5001

Marine Sciences Collection
Mayaguez Campus Library
University of Puerto Rico
Mayagues, Puerto Rico 00708

50272-101

REPORT DOCUMENTATION PAGE	1. REPORT NO. WHOI-89-48	2.	3. Recipient's Accession No.

4. Title and Subtitle Calculation of 3-Dimensional Synthetic Seismograms on the Connection Machine	5. Report Date October, 1989
	6.
7. Author(s) J.M. Allen and D.R. Burns	8. Performing Organization Rept. No. WHOI-89-48
9. Performing Organization Name and Address The Woods Hole Oceanographic Institution Woods Hole, Massachusetts 02543	10. Project/Task/Work Unit No.
	11. Contract(C) or Grant(G) No. (C) N00014-87-K-0007 N00014-89-J-1012 (G)
12. Sponsoring Organization Name and Address Funding was provided by the Office of Naval Research	13. Type of Report & Period Covered Technical Report
	14.

15. Supplementary Notes

This report should be cited as: Woods Hole Oceanog. Inst. Tech. Rept., WHOI-89-48

16. Abstract (Limit: 200 words)

A three dimensional, second order finite difference method was used to create synthetic seismograms for elastic wave propagation in heterogeneous media. These synthetic seismograms are used to model rough seafloor, the shallow crust, or complex structural and stratigraphic settings with strong lateral heterogeneities. The finite difference method is preferred because it allows models of any complexity to be generated and includes all multiple scattering, wave conversion and diffraction effects. The method uses a fully staggered grid as developed by Virieux (1986). Wavefront snapshots and time series output allow the scattering and focussing of different wave modes with direction to be visualized.

The extensive calculations required for realistic size models stretches the resources of serial computers like the VAX 8800. On the Connection Machine, a massively parallel computer, the finite difference grid can be directly mapped onto the virtual processors, reducing the nested time and space loops in the serial code to a single time loop. As a result, the computation time is reduced dramatically.

17. Document Analysis a. Descriptors

1. 3-D synthetic seismograms
2. parallel computing
3. acoustic modeling

b. Identifiers/Open-Ended Terms

c. COSATI Field/Group

18. Availability Statement Approved for publication; distribution unlimited.	19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages 79
	20. Security Class (This Page)	22. Price